Bài giảng hệ thống nhúng

Biên tập bởi:

Khoa CNTT ĐH Công nghệ Giao thông vận tải

MỤC LỤC

Tổng quan hệ thống nhúng

Giới thiệu môn học

Module này cung cấp cho người học các kiến thức cơ bản về hệ thống nhúng, Nội dung chính bao gồm: Giới thiệu chung về các hệ thống nhúng; Kiến trúc phần cứng hệ thống nhúng; Hệ điều hành nhúng, phần mềm nhúng.

Sau khi hoàn thành module này, người học có khả năng:

- Giải thích được một số các khái niệm liên quan đến hệ thống nhúng, hệ điều hành nhúng và phần mềm nhúng.
- Phân tích được các đặc điểm cấu trúc phần cứng, hệ điều hành và phần mềm cho các hệ thống nhúng.
- Ứng dụng trong thiết kế và phát triển phần mềm cho hệ thống nhúng đơn giản.
- Rèn luyện khả năng tự nghiên cứu, khả năng làm việc theo nhóm.

Để học tốt môn học này mỗi người học phải tự xây dựng cho mình một phương pháp học thích hợp. Nhưng phương pháp chung để học môn học này là người học phải hiểu thật kỹ các phần lý thuyết cơ bản từ đó tìm hiểu các phần kiến thức mở rộng.

Hệ thống nhúng là gì?

Hệ thống nhúng (Embedded system) là một thuật ngữ để chỉ một hệ thống có khả năng tự trị được nhúng vào trong một môi trường hay một hệ thống mẹ. Đó là các hệ thống tích hợp cả phần cứng và phần phềm để thực hiện một hoặc một nhóm chức năng chuyên biệt cụ thể .

Hệ thống nhúng (HTN) thường được thiết kế để thực hiện một chức năng chuyên biệt nào đó. Khác với các máy tính đa chức năng, chẳng hạn như máy tính cá nhân, một hệ thống nhúng chỉ thực hiện một hoặc một vài chức năng nhất định, thường đi kèm với những yêu cầu cụ thể và bao gồm một số thiết bị máy móc và phần cứng chuyên dụng mà ta không tìm thấy trong một máy tính đa năng nói chung. Vì hệ thống chỉ được xây dựng cho một số nhiệm vụ nhất định nên các nhà thiết kế có thể tối ưu hóa nó nhằm giảm thiểu kích thước và chi phí sản xuất. Các hệ thống nhúng thường được sản xuất hàng loạt với số lượng lớn. HTN rất đa dạng, phong phú về chủng loại. Đó có thể là những thiết bị cầm tay nhỏ gọn như đồng hồ kĩ thuật số và máy chơi nhạc MP3, hoặc những sản phẩm lớn như đèn giao thông, bộ kiểm soát trong nhà máy hoặc hệ thống kiểm soát các máy năng lượng hạt nhân. Xét về độ phức tạp, hệ thống nhúng có thể rất đơn giản với một vi điều khiển hoặc rất phức tạp với nhiều đơn vị, các thiết bị ngoại vi và mạng lưới được nằm gọn trong một lớp vỏ máy lớn.

Các thiết bị PDA hoặc máy tính cầm tay cũng có một số đặc điểm tương tự với hệ thống nhúng như các hệ điều hành hoặc vi xử lý điều khiển chúng nhưng các thiết bị này không phải là hệ thống nhúng thật sự bởi chúng là các thiết bị đa năng, cho phép sử dụng nhiều ứng dụng và kết nối đến nhiều thiết bị ngoại vi.

Lịch sử phát triển của hệ thống nhúng

Hệ thống nhúng đầu tiên là *Apollo Guidance Computer*(Máy tính dẫn đường Apollo) được phát triển bởi *Charles Stark Draper* tại phòng thí nghiệm của trường đại học *MIT*năm 1960. Hệ thống nhúng được sản xuất hàng loạt đầu tiên là máy hướng dẫn cho tên lửa quân sự vào năm 1961. Nó là máy hướng dẫn *Autonetics D-17*, được xây dựng sử dụng những bóng bán dẫn và một đĩa cứng để duy trì bộ nhớ. Khi *Minuteman II* được đưa vào sản xuất năm 1996, *AutoneticsD-17* đã được thay thế với một máy tính mới sử dụng mạch tích hợp. Tính năng thiết kế chủ yếu của máy tính *Minuteman II* là nó đưa ra thuật toán có thể lập trình lại sau đó để làm cho tên lửa chính xác hơn, và máy tính có thể kiểm tra tên lửa, giảm trọng lượng của cáp điện và đầu nối điện.

Từ những ứng dụng đầu tiên vào những năm 1960, các hệ thống nhúng phát triển mạnh mẽ về khả năng xử lý. Bộ vi xử lý đầu tiên hướng đến người tiêu dùng là *Intel 4004,* được phát minh phục vụ máy tính điện tử và những hệ thống nhỏ khác. Tuy nhiên nó vẫn cần các chip nhớ ngoài và những hỗ trợ khác. Vào những năm cuối 1970, những bộ xử lý 8 bit đã được sản xuất, nhưng nhìn chung chúng vẫn cần đến những chip nhớ bên ngoài.

Vào giữa thập niên 80, kỹ thuật mạch tích hợp đã đạt trình độ cao dẫn đến nhiều thành phần có thể đưa vào một chip xử lý. Các bộ vi xử lý được gọi là các vi điều khiển và được chấp nhận rộng rãi. Với giá cả thấp, các vi điều khiển đã trở nên rất hấp dẫn để xây dựng các hệ thống chuyên dụng. Đã có một sự bùng nổ về số lượng các hệ thống nhúng trong tất cả các lĩnh vực thị trường và số các nhà đầu tư sản xuất theo hướng này. Ví dụ, rất nhiều chip xử lý đặc biệt xuất hiện với nhiều giao diện lập trình hơn là kiểu song song truyền thống để kết nối các vi xử lý. Vào cuối những năm 80, các hệ thống nhúng đã trở nên phổ biến trong hầu hết các thiết bị điện tử và khuynh hướng này vẫn còn tiếp tục cho đến nay.

Xu hướng phát triển của các hệ thống nhúng

Sau máy tính lớn (mainframe), PC và Internet thì hệ thống nhúng đang là làn sóng đổi mới thứ 3 trong công nghệ thông tin và truyền thông.

Xu hướng phát triển của các hệ thống nhúng hiện nay là:

- Phần mềm ngày càng chiếm tỷ trọng cao và đã trở thành một thành phần cấu tạo nên thiết bị bình đẳng như các phần cơ khí, linh kiện điện tử, linh kiện quang học…
- Các hệ nhúng ngày càng phức tạp hơn đáp ứng các yêu cầu khắt khe về thời gian thực, tiêu ít năng lượng và hoạt động tin cậy ổn định hơn.
- Các hệ nhúng ngày càng có độ mềm dẻo cao đáp ứng các yêu cầu nhanh chóng đưa sản phẩm ra thương trường, có khả năng bảo trì từ xa, có tính cá nhân cao.
- Các hệ nhúng ngày càng có khả năng hội thoại cao, có khả năng kết nối mạng và hội thoại với người sử dụng.
- Các hệ nhúng ngày càng có tính thích nghi, tự tổ chức cao có khả năng tái cấu hình như một thực thể, một tác nhân.
- Các hệ nhúng ngày càng có khả năng tiếp nhận năng lượng từ nhiều nguồn khác nhau (ánh sáng, rung động, điện từ trường, sinh học….) để tạo nên các hệ thống tự tiếp nhận năng lượng trong quá trình hoạt động.

Những thách thức và vấn đề còn tồn tại với hệ thống nhúng

Hệ thống nhúng hiện nay còn phải đối mặt với các vấn đề sau:

- Độ phức tạp của sự liên kết đa ngành phối hợp cứng - mềm.Độ phức tạp của hệ thống tăng cao do nó kết hợp nhiều lĩnh vực đa ngành, kết hợp phần cứng - mềm, trong khi các phương pháp thiết kế và kiểm tra chưa chín muồi. Khoảng cách giữa lý thuyết và thực hành lớn và còn thiếu các phương pháp và lý thuyết hoàn chỉnh cho khảo sát phân tích toàn cục các hệ nhúng.
- Thiếu phương pháp tích hợp tối ưu giữa các thành phần tạo nên hệ nhúng bao gồm lý thuyết điều khiển tự động, thiết kế máy, công nghệ phần mềm, điện tử, vi xử lý, các công nghệ hỗ trợ khác.
- Thách thức đối với độ tin cậy và tính mở của hệ thống: Do hệ thống nhúng thường phải hội thoại với môi trường xung quanh nên nhiều khi gặp những tình huống không được thiết kế trước dễ dẫn đến hệ thống bị loạn. Trong quá trình hoạt động một số phần mềm thường phải chỉnh lại và thay đổi nên hệ thống phần mềm có thể không kiểm soát được. Đối với hệ thống mở, các hãng thứ 3 đưa các module mới, thành phần mới vào cũng có thể gây nên sự hoạt động thiếu tin cậy.

Các đặc điểm của hệ thống nhúng

Hệ thống nhúng thường có một số đặc điểm chung như sau:

- Các hệ thống nhúng được thiết kế để thực hiện một số nhiệm vụ chuyên dụng chứ không phải đóng vai trò là các hệ thống máy tính đa chức năng. Một số hệ thống đòi hỏi ràng buộc về tính hoạt động thời gian thực để đảm bảo độ an toàn

và tính ứng dụng. Một số hệ thống không đòi hỏi hoặc ràng buộc chặt chẽ, cho phép đơn giản hóa hệ thống phần cứng để giảm thiểu chi phí sản xuất.
- Một hệ thống nhúng thường không phải là một khối riêng biệt mà là một hệ thống phức tạp nằm trong thiết bị mà nó điều khiển.
- Phần mềm được viết cho các hệ thống nhúng được gọi là *firmware* và được lưu trữ trong các chip bộ nhớ chỉ đọc (ROM - Read Only Memory) hoặc bộ nhớ flash chứ không phải là trong một ổ đĩa. Phần mềm thường chạy với số tài nguyên phần cứng hạn chế: không có bàn phím, màn hình hoặc có nhưng với kích thước nhỏ, bộ nhớ hạn chế.

Sau đây, chúng ta sẽ đi sâu, xem xét cụ thể đặc điểm của các thành phần của hệ thống nhúng.

Giao diện

Các hệ thống nhúng có thể không có giao diện (đối với những hệ thống đơn nhiệm) hoặc có đầy đủ giao diện giao tiếp với người dùng tương tự như các hệ điều hành trong các thiết bị để bàn. Đối với các hệ thống đơn giản, thiết bị nhúng sử dụng nút bấm, đèn LED và hiển thị chữ cỡ nhỏ hoặc chỉ hiển thị số, thường đi kèm với một hệ thống menu đơn giản.

Còn trong một hệ thống phức tạp hơn, một màn hình đồ họa, cảm ứng hoặc có các nút bấm ở lề màn hình cho phép thực hiện các thao tác phức tạp mà tối thiểu hóa được khoảng không gian cần sử dụng. Ý nghĩa của các nút bấm có thể thay đổi theo màn hình và các lựa chọn. Các hệ thống nhúng thường có một màn hình với một nút bấm dạng cần điều khiển (*joystick button*). Sự phát triển mạnh mẽ của mạng toàn cầu đã mang đến cho những nhà thiết kế hệ nhúng một lựa chọn mới là sử dụng một giao diện web thông qua việc kết nối mạng. Điều này có thể giúp tránh được chi phí cho những màn hình phức tạp nhưng đồng thời vẫn cung cấp khả năng hiển thị và nhập liệu phức tạp khi cần đến, thông qua một máy tính khác. Điều này là hết sức hữu dụng đối với các thiết bị điều khiển từ xa, cài đặt vĩnh viễn. Ví dụ, các router là các thiết bị đã ứng dụng tiện ích này.

Kiến trúc CPU

Các bộ xử lý trong hệ thống nhúng có thể được chia thành hai loại: Vi xử lý và vi điều khiển. Các vi điều khiển thường có các thiết bị ngoại vi được tích hợp trên chip nhằm giảm kích thước của hệ thống. Có rất nhiều loại kiến trúc CPU được sử dụng trong thiết kế hệ nhúng như ARM, MIPS, Coldfire/68k, PowerPC, x86, PIC, 8051, Atmel AVR… Điều này trái ngược với các loại máy tính để bàn, thường bị hạn chế với một vài kiến trúc máy tính nhất định. Các hệ thống nhúng có kích thước nhỏ và được thiết kế để hoạt động trong môi trường công nghiệp thường lựa chọn PC/104 và PC/104++ làm nền tảng. Những hệ thống này thường sử dụng DOS, Linux hoặc các hệ điều hành nhúng thời gian thực như QNX hay VxWorks. Còn các hệ thống nhúng có kích thước rất lớn

thường sử dụng một cấu hình thông dụng là *hệ thống on chip* (*System on a chip – SoC*), một bảng mạch tích hợp cho một ứng dụng cụ thể (An Application Specific Integrated Circuit – ASIC). Sau đó nhân CPU thêm vào như một phần của thiết kế chip. Một chiến lược tương tự là sử dụng FPGA (field-programmable gate array) và lập trình cho nó với những thành phần nguyên lý thiết kế bao gồm cả CPU.

Thiết bị ngoại vi

Hệ thống nhúng giao tiếp với bên ngoài thông qua các thiết bị ngoại vi, ví dụ như:

- Serial Communication Interfaces (SCI): RS-232, RS-422, RS-485.
- Universal Serial Bus (USB).
- Networks: Controller Area Network, LonWorks.
- Bộ định thời: PLL(s), Capture/Compare và Time Processing Units.
- Discrete IO: General Purpose Input/Output (GPIO).

Công cụ phát triển

Tương tự như các sản phẩm phần mềm khác, phần mềm hệ thống nhúng cũng được phát triển nhờ việc sử dụng các trình biên dịch (compilers), chương trình dịch hợp ngữ (assembler) hoặc các công cụ gỡ rối (debuggers). Tuy nhiên, các nhà thiết kế hệ thống nhúng có thể sử dụng một số công cụ chuyên dụng như:

- Bộ gỡ rối mạch hoặc các chương trình mô phỏng (*emulator*).
- Tiện ích để thêm các giá trị checksum hoặc CRC vào chương trình, giúp hệ thống nhúng có thể kiểm tra tính hợp lệ của chương trình đó.
- Đối với các hệ thống xử lý tín hiệu số, người phát triển hệ thống có thể sử dụng phần mềm workbench như MathCad hoặc Mathematica để mô phỏng các phép toán.
- Các trình biên dịch và trình liên kết (linker) chuyên dụng được sử dụng để tối ưu hóa một thiết bị phần cứng.
- Một hệ thống nhúng có thể có ngôn ngữ lập trình và công cụ thiết kế riêng của nó hoặc sử dụng và cải tiến từ một ngôn ngữ đã có sẵn.

Các công cụ phần mềm có thể được tạo ra bởi các công ty phần mềm chuyên dụng về hệ thống nhúng hoặc chuyển đổi từ các công cụ phát triển phần mềm GNU. Đôi khi, các công cụ phát triển dành cho máy tính cá nhân cũng được sử dụng nếu bộ xử lý của hệ thống nhúng đó gần giống với bộ xử lý của một máy PC thông dụng.

Độ tin cậy

Các hệ thống nhúng thường nằm trong các cỗ máy được kỳ vọng là sẽ chạy hàng năm trời liên tục mà không bị lỗi hoặc có thể khôi phục hệ thống khi gặp lỗi. Vì thế, các phần

mềm hệ thống nhúng được phát triển và kiểm thử một cách cẩn thận hơn là phần mềm cho máy tính cá nhân. Ngoài ra, các thiết bị rời không đáng tin cậy như ổ đĩa, công tắc hoặc nút bấm thường bị hạn chế sử dụng. Việc khôi phục hệ thống khi gặp lỗi có thể được thực hiện bằng cách sử dụng các kỹ thuật như *watchdog timer* – nếu phần mềm không đều đặn nhận được các tín hiệu *watchdog* định kì thì hệ thống sẽ bị khởi động lại.

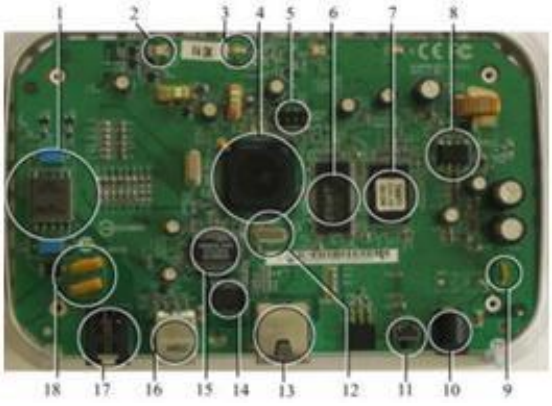
Một số vấn đề cụ thể về độ tin cậy như:

- Hệ thống không thể ngừng để sửa chữa một cách an toàn, ví dụ như ở các hệ thống không gian, hệ thống dây cáp dưới đáy biển, các đèn hiệu dẫn đường… Giải pháp đưa ra là chuyển sang sử dụng các hệ thống con dự trữ hoặc các phần mềm cung cấp một phần chức năng.
- Hệ thống phải được chạy liên tục vì tính an toàn, ví dụ như các thiết bị dẫn đường máy bay, thiết bị kiểm soát độ an toàn trong các nhà máy hóa chất… Giải pháp đưa ra là lựa chọn backup hệ thống.
- Nếu hệ thống ngừng hoạt động sẽ gây tổn thất rất nhiều tiền của ví dụ như các dịch vụ buôn bán tự động, hệ thống chuyển tiền, hệ thống kiểm soát trong các nhà máy …

Một số ví dụ về hệ thống nhúng

Quanh ta có rất nhiều sản phẩm nhúng như lò vi sóng, nồi cơm điện, điều hoà, điện thoại di động, ô tô, máy bay, tàu thuỷ, các đầu đo cơ cấu chấp hành thông minh …. Ta có thể thấy hiện nay hệ thống nhúng có mặt ở mọi lúc mọi nơi trong cuộc sống của chúng ta.

Các máy trả lời tự động, các thiết bị y tế, máy in, hệ thống dẫn đường trong không lưu đều có tích hợp các hệ thống nhúng.



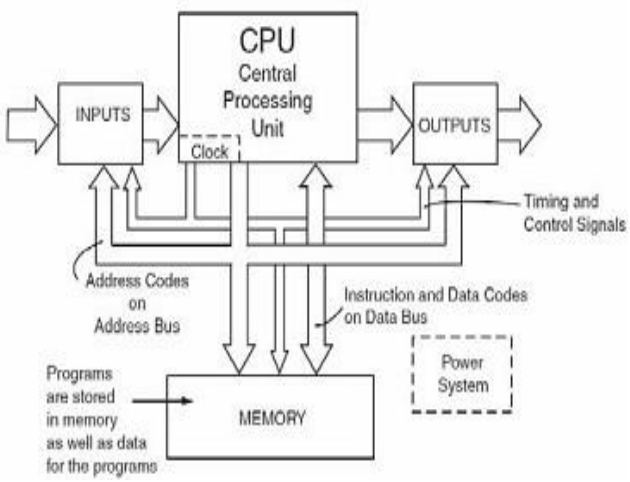
Cấu trúc bên trong Router

Router là một ví dụ của hệ thống nhúng. Các hệ thống nhúng trong mô hình Router bao gồm: *Microprocessor* (4), *RAM* (6), và *Flash memory*(7).

Các thiết bị trên các tàu vũ trụ được tích hợp rất nhiều các hệ thống nhúng.

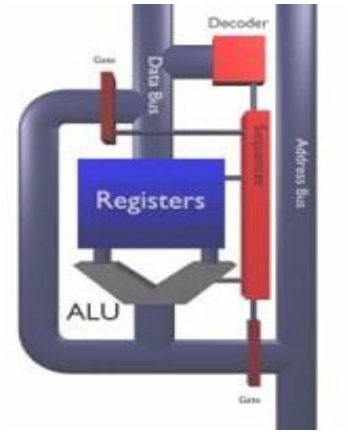
Tàu thăm dò Sao Hỏa

Các thành phần cơ bản trong kiến trúc phần cứng Hệ thống nhúng



Kiến trúc điển hình của các chip VXL/VĐK nhúng

Đơn vị xử lý trung tâm CPU



Kiến trúc CPU

Người ta vẫn biết tới phần lõi xử lý của các bộ vi xử lý (VXL) là đơn vị xử lý trung tâm CPU (Central Processing Unit) đóng vai trò như bộ não chịu trách nhiệm thực thi các phép tính và thực hiện các lệnh. Phần chính của CPU đảm nhận chức năng này là đơn vị logic toán học (ALU - Arthimetic Logic Unit). Ngoài ra để hỗ trợ hoạt động cho ALU còn thêm một số thành phần khác như bộ giải mã (*decoder*), bộ tuần tự (*Sequencer*) và các thanh ghi.

Bộ giải mã chuyển đổi (thông dịch) các lệnh lưu trữ ở trong bộ mã chương trình thành các mã mà ALU có thể hiểu được và thực thi. Bộ tuần tự có nhiệm vụ quản lý dòng dữ liệu trao đổi qua bus dữ liệu của VXL. Các thanh ghi được sử dụng để CPU lưu trữ tạm thời các dữ liệu chính cho việc thực thi các lệnh và chúng có thể thay đổi nội dung trong quá trình hoạt động của ALU. Hầu hết các thanh ghi của VXL đều là các bộ nhớ được tham chiếu (mapped) và hội nhập với khu vực bộ nhớ và có thể được sử dụng như bất kỳ khu vực nhớ khác.

Các thanh ghi có chức năng lưu trữ trạng thái của CPU. Nếu các nội dung của bộ nhớ VXL và các nội dung của các thanh ghi tại một thời điểm nào đó được lữu giữ đầy đủ thì hoàn toàn có thể tạm dừng thực hiện phần chương trình hiện tại trong một khoảng thời gian bất kỳ và có thể trở lại trạng thái của CPU trước đó. Thực tế số lượng các thanh ghi và tên gọi của chúng cũng khác nhau trong các họ VXL/VĐK và thường do chính các nhà chế tạo qui định, nhưng về cơ bản chúng đều có chung các chức năng như đã nêu.

Khi thứ tự byte trong bộ nhớ đã được xác định thì người thiết kế phần cứng phải thực hiện một số quyết định xem CPU sẽ lưu dữ liệu đó như thế nào. Cơ chế này cũng khác nhau tuỳ theo kiến trúc tập lệnh được áp dụng. Có ba loại hình cơ bản:

Kiến trúc ngăn xếp.

Kiến trúc bộ tích luỹ.

Kiến trúc thanh ghi mục đích chung.

Kiến trúc ngăn xếp sử dụng ngăn xếp để thực hiện lệnh và các toán tử nhận được từ đỉnh ngăn xếp. Mặc dù cơ chế này hỗ trợ mật độ mã tốt và mô hình đơn giản cho việc đánh giá cách thể hiện chương trình nhưng ngăn xếp không thể hỗ trợ khả năng truy nhập ngẫu nhiên và hạn chế hiệu suất thực hiện lệnh.

Kiến trúc bộ tích luỹ với lệnh một toán tử ngầm mặc định chứa trong thanh ghi tích luỹ có thể giảm được độ phức tạp bên trong của cấu trúc CPU và cho phép cấu thành lệnh rất nhỏ gọn. Nhưng thanh ghi tích luỹ chỉ là nơi chứa dữ liệu tạm thời nên giao thông bộ nhớ rất lớn.

Kiến trúc thanh ghi mục đích chung sử dụng các tập thanh ghi mục đích chung và được đón nhận như mô hình của các hệ thống CPU mới, hiện đại. Các tập thanh ghi đó nhanh hơn bộ nhớ thường và dễ dàng cho bộ biên dịch xử lý thực thi và có thể được sử dụng một cách hiệu quả. Hơn nữa giá thành phần cứng ngày càng có xu thế giảm đáng kể và tập thanh ghi có thể tăng nhanh. Nếu cơ chế truy nhập bộ nhớ nhanh thì kiến trúc dựa trên ngăn xếp có thể là sự lựa chọn lý tưởng, còn nếu truy nhập bộ nhớ chậm thì kiến trúc thanh ghi sẽ là sự lựa chọn phù hợp nhất.

Một số thanh ghi với chức năng điển hình thường được sử dụng trong các kiến trúc CPU như sau:

Thanh ghi con trỏ ngăn xếp (stack pointer): Thanh ghi này lưu giữ địa chỉ tiếp theo của ngăn xếp. Theo nguyên lý giá trị địa chỉ chứa trong thanh ghi con trỏ ngăn xếp sẽ giảm nếu dữ liệu được lưu thêm vào ngăn xếp và sẽ tăng khi dữ liệu được lấy ra khỏi ngăn xếp.

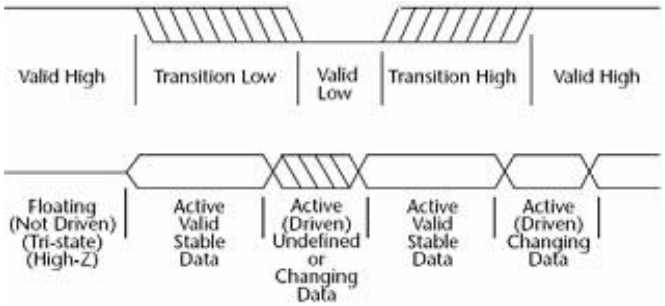
Thanh ghi chỉ số (index register): Thanh ghi chỉ số được sử dụng để lưu địa chỉ khi mode địa chỉ được sử dụng. Nó còn được biết tới với tên gọi là thanh ghi con trỏ hay thanh ghi lựa chọn tệp (Microchip).

Thanh ghi địa chỉ lệnh/Bộ đếm chương trình (Program Counter): Một trong những thanh ghi quan trọng nhất của CPU là thanh ghi bộ đếm chương trình. Thanh ghi bộ đếm chương trình lưu địa chỉ lệnh tiếp theo của chương trình sẽ được CPU xử lý. Mỗi khi lệnh được trỏ tới và được CPU xử lý thì nội dung giá trị của thanh ghi bộ đếm chương trình sẽ tăng lên một. Chương trình sẽ kết thúc khi thanh ghi PC có giá trị bằng địa chỉ cuối cùng của chương trình nằm trong bộ nhớ chương trình.

Thanh ghi tích lũy (Accumulator): Thanh ghi tích lũy là một thanh ghi giao tiếp trực tiếp với ALU, được sử dụng để lưu giữ các toán tử hoặc kết quả của một phép toán trong quá trình hoạt động của ALU.

Xung nhịp và trạng thái tín hiệu

Trong VXL và các vi mạch số nói chung, hoạt động của hệ thống được thực hiện đồng bộ hoặc dị bộ theo các xung nhịp chuẩn. Các nhịp đó được lấy trực tiếp hoặc gián tiếp từ một nguồn xung chuẩn thường là các mạch tạo xung. Để mô tả hoạt động của hệ thống, các tín hiệu dữ liệu và điều khiển thường được mô tả trạng thái theo giản đồ thời gian và mức tín hiệu như được chỉ ra trong Hình 2.3.



Mô tả và trạng thái tín hiệu hoạt động trong VXL

Mục đích của việc mô tả trạng thái tín hiệu theo giản đồ thời gian và mức tín hiệu là để phân tích và xác định chuỗi sự kiện hoạt động chi tiết trong mỗi chu kỳ bus. Nhờ việc

mô tả này chúng ta có thể xem xét đến khả năng đáp ứng thời gian của các sự kiện thực thi trong hệ thống và thời gian cần thiết để thực thi hoạt động tuần tự cũng như là khả năng tương thích khi có sự hoạt động phối hợp giữa các thiết bị ghép nối hay mở rộng trong hệ thống. Thông thường thông tin về các nhịp thời gian hoạt động cũng như đặc tính kỹ thuật chi tiết được cung cấp hoặc qui định bởi các nhà chế tạo.

Một số đặc trưng về thời gian của các trạng thái hoạt động cơ bản của các tín hiệu hệ thống gồm có như sau:

Thời gian tăng hoặc giảm.

Thời gian trễ lan truyền tín hiệu.

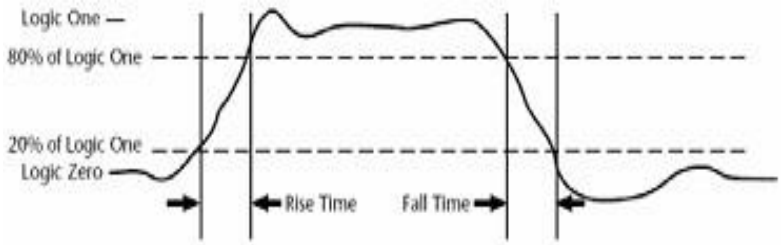
Thời gian thiết lập và lưu giữ.

Trễ cấm hoạt động và trạng thái treo (*Tri-State*).

Độ rộng xung.

Tần số nhịp xung hoạt động.

- *Thời gian tăng hoặc giảm*:

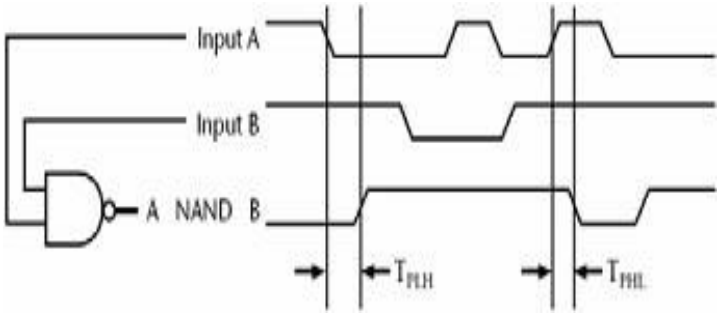


Mô tả trạng thái tín hiệu tăng và giảm

Thời gian tăng được định nghĩa là khoảng thời gian để tín hiệu tăng từ 20% đến 80% mức tín hiệu cần thiết. *Thời gian giảm* là khoảng thời gian để tín hiệu giảm từ 80% đến 20% mức tín hiệu cần thiết.

- *Thời gian trễ lan truyền tín hiệu:*

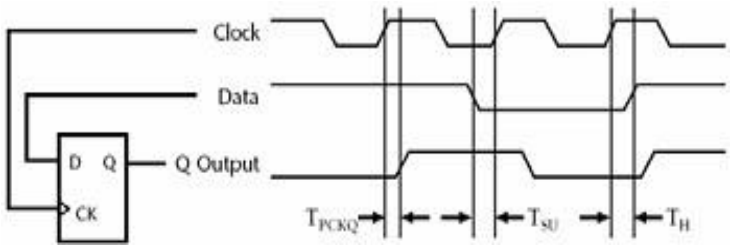
Là khoảng thời gian tính từ khi thay đổi tín hiệu vào cho tới khi có sự thay đổi tín hiệu ở đầu ra. Đặc tính này thường do cấu tạo và khả năng truyền dẫn tín hiệu vật lý trong hệ thống tín hiệu.



Mô tả trạng thái và độ trễ lan truyền tín hiệu

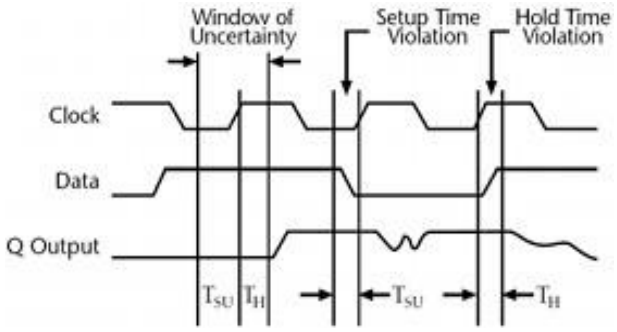
Thời gian thiết lập và lưu giữ:

Khoảng thời gian cần thiết để tín hiệu trích mẫu đạt tới một trạng thái ổn định trước khi nhịp xung chuẩn đồng hồ thay đổi được gọi là thời gian thiết lập. Thời gian lưu giữ là khoảng thời gian cần thiết để duy trì tín hiệu trích mẫu ổn định sau khi xung nhịp chuẩn đồng hồ thay đổi. Thực chất khoảng thời gian thiết lập và thời gian lưu giữ là cần thiết để đảm bảo tín hiệu được ghi nhận chính xác và ổn định trong quá trình hoạt động và chuyển mức trạng thái. Giản đồ thời gian trong Hình 2.6: Thời gian thiết lập và lưu giữ minh họa thời gian thiết lập và lưu giữ trong hoạt động của Triger D.



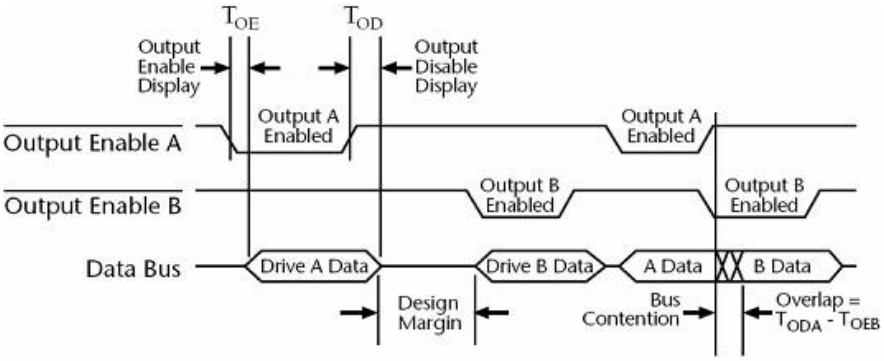
Thời gian thiết lập và lưu trữ

Trong trường hợp hoạt động chuyển trạng thái tín hiệu không đồng bộ và không đảm bảo được thời gian thiết lập và lưu giữ sẽ có thể dẫn đến sự mất ổn định hay không xác định mức tín hiệu trong hệ thống. Hiện tượng này được biết tới với tên gọi là *metastabilit*. Để minh họa cho hiện tượng này trong Hình 2.7 mô tả hoạt động lỗi của một Triger khi các mức tín hiệu vào không thỏa mãn yêu cầu về thời thiết lập và lưu giữ.



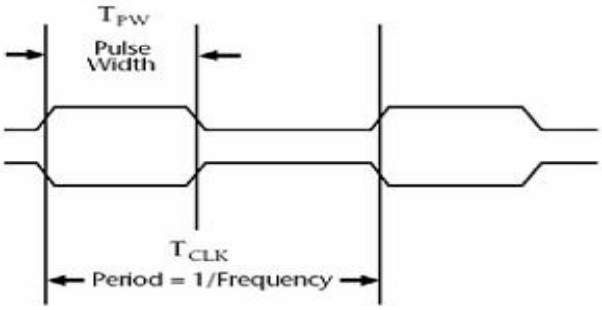
Hiện tượng Metastabilit trong hoạt động của Triger D

Chu kì tín hiệu 3 trạng thái và contention:



Mô tả chu kì tín hiệu 3 trạng thái và Contention

Độ rộng xung và tần số nhịp xung chuẩn:



Độ rộng và tần số xung nhịp chuẩn

Bus địa chỉ, dữ liệu và điều khiển

Bus địa chỉ:

Bus địa chỉ là các đường dẫn tín hiệu logic một chiều để truyền địa chỉ tham chiếu tới các khu vực bộ nhớ và chỉ ra dữ liệu được lưu giữ ở đâu trong không gian bộ nhớ. Trong qúa trình hoạt động CPU sẽ điều khiển bus địa chỉ để truyền dữ liệu giữa các khu vực bộ nhớ và CPU. Các địa chỉ thông thường tham chiếu tới các khu vực bộ nhớ hoặc các khu vực vào ra, hoặc ngoại vi. Dữ liệu được lưu ở các khu vực đó thường là 8bit (1 byte), 16bit, hoặc 32bit tùy thuộc vào cấu trúc từng loại vi xử lý/vi điều khiển. Hầu hết các vi điều khiển thường đánh địa chỉ dữ liệu theo khối 8bit. Các loại vi xử lý 8bit, 16bit và 32bit nói chung cũng đều có thể làm việc trao đổi với kiểu dữ liệu 8bit và 16bit.

Chúng ta vẫn thường được biết tới khái niệm địa chỉ truy nhập trực tiếp, đó là khả năng CPU có thể tham chiếu và truy nhập tới trong một chu kỳ bus. Nếu vi xử lý có N bit địa chỉ tức là nó có thể đánh địa chỉ được 2N khu vực mà CPU có thể tham chiếu trực tiếp tới. Qui ước các khu vực được đánh địa chỉ bắt đầu từ địa chỉ 0 và tăng dần đến 2**N**-1. Hiện nay các vi xử lý và vi điều khiển nói chung chủ yếu vẫn sử dụng phổ biến các bus

dữ liệu có độ rộng là 16, 20, 24, hoặc 32bit. Nếu đánh địa chỉ theo byte thì một vi xử lý 16bit có thể đánh địa chỉ được 216 khu vực bộ nhớ tức là 65,536 byte = 64Kbyte. Tuy nhiên có một số khu vực bộ nhớ mà CPU không thể truy nhập trực tiếp tới tức là phải sử dụng nhiều nhịp bus để truy nhập, thông thường phải kết hợp với việc điều khiển phần mềm. Kỹ thuật này chủ yếu được sử dụng để mở rộng bộ nhớ và thường được biết tới với khái niệm đánh địa chỉ trang nhớ khi nhu cầu đánh địa chỉ khu vực nhớ vượt quá phạm vi có thể đánh địa chỉ truy nhập trực tiếp.

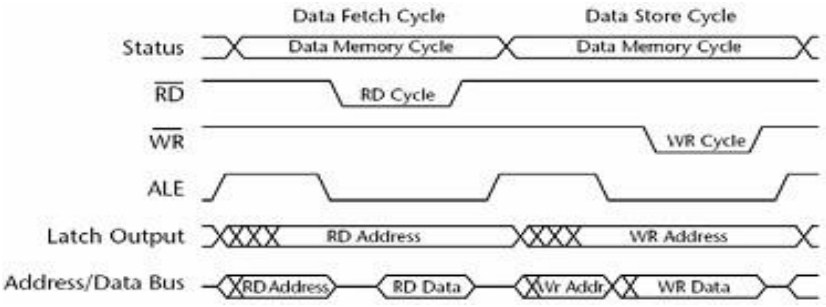
Ví dụ: CPU 80286 có 24bit địa chỉ sẽ cho phép đánh địa chỉ trực tiếp cho 224 byte nhớ. CPU 80386 và các loại vi xử lý mạnh hơn có không gian địa chỉ 32bit sẽ có thể đánh được tới 232 byte địa chỉ trực tiếp.

Bus dữ liệu

Bus dữ liệu là các kênh truyền tải thông tin theo hai chiều giữa CPU và bộ nhớ hoặc các thiết bị ngoại vi vào ra. Bus dữ liệu được điều khiển bởi CPU để đọc hoặc viết các dữ liệu hoặc mã lệnh thực thi trong quá trình hoạt động của CPU. Độ rộng của bus dữ liệu nói chung sẽ xác định được lượng dữ liệu có thể truyền và trao đổi trên bus. Tốc độ truyền hay trao đổi dữ liệu thường được tính theo đơn vị là [byte/s]. Số lượng đường bit dữ liệu sẽ cho phép xác định được số lượng bit có thể lưu trữ trong mỗi khu vực tham chiếu trực tiếp. Nếu một bus dữ liệu có khả năng thực hiện một lần truyền trong 1 μs, thì bus dữ liệu 8bit sẽ có băng thông là 1Mbyte/s, bus 16bit sẽ có băng thông là 2Mbyte/s và bus 32bit sẽ có băng thông là 4Mbyte/s. Trong trường hợp bus dữ liệu 8bit với chu kỳ bus là T=1μs (tức là sẽ truyền được 1byte/1chu kỳ) thì sẽ truyền được 1 Mbyte trong 1s hay 2Mbyte trong 2s.

Bus điều khiển

Bus điều khiển phục vụ truyền tải các thông tin dữ liệu để điều khiển hoạt động của hệ thống. Thông thường các dữ liệu điều khiển bao gồm các tín hiệu chu kỳ để đồng bộ các nhịp chuyển động và hoạt động của hệ thống. Bus điều khiển thường được điều khiển bởi CPU để đồng bộ hóa nhịp hoạt động và dữ liệu trao đổi trên các bus. Trong trường hợp vi xử lý sử dụng dồn kênh bus dữ liệu và bus địa chỉ tức là một phần hoặc toàn bộ bus dữ liệu sẽ được sử dụng chung chia sẻ với bus địa chỉ thì cần một tín hiệu điều khiển để phân nhịp truy nhập cho phép chốt lưu trữ thông tin địa chỉ mỗi khi bắt đầu một chu kỳ truyền. Một ví dụ về các chu kỳ bus và sự đồng bộ của chúng trong hoạt động của hệ thống bus địa chỉ và dữ liệu dồn kênh được chỉ ra trong Hình 2.10: Đây là hoạt động điển hình trong họ vi điều khiển 8051 và nhiều loại tương tự.

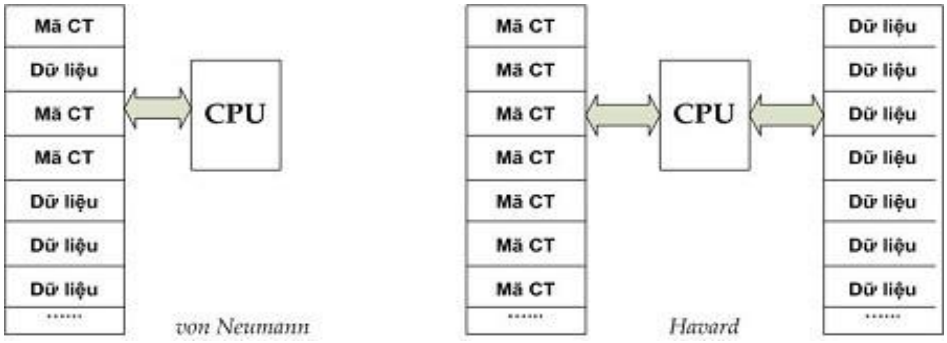


Chu kì hoạt động bus dồn kênh

Bộ nhớ

Kiến trúc bộ nhớ:

Kiến trúc bộ nhớ được chia ra làm hai loại chính và được áp dụng rộng rãi trong hầu hết các Chip xử lý nhúng hiện nay là kiến trúc bộ nhớ *von Neumann* và *Havard.* Trong kiến trúc *von Neumann* không phân biệt vùng chứa dữ liệu và mã chương trình. Cả chương trình và dữ liệu đều được truy nhập theo cùng một đường. Điều này cho phép đưa dữ liệu vào vùng mã chương trình ROM, và cũng có thể lưu mã chương trình vào vùng dữ liệu RAM và thực hiện từ đó.



Kiến trúc bộ nhớ von Neumann và Havard

Kiến trúc *Havard* tách/phân biệt vùng lưu mã chương trình và dữ liệu. Mã chương trình chỉ có thể được lưu và thực hiện trong vùng chứa ROM và dữ liệu cũng chỉ có thể lưu và trao đổi trong vùng RAM. Hầu hết các vi xử lý nhúng ngày nay sử dụng kiến trúc bộ nhớ Havard hoặc kiến trúc Havard mở rộng (tức là bộ nhớ chương trình và dữ liệu tách biệt nhưng vẫn cho phép khả năng hạn chế để lấy dữ liệu ra từ vùng mã chương trình). Trong kiến trúc bộ nhớ Havard mở rộng thường sử dụng một số lượng nhỏ các con trỏ để lấy dữ liệu từ vùng mã chương trình theo cách nhúng vào trong các lệnh tức thời. Một số Chip vi điều khiển nhúng tiêu biểu hiện nay sử dụng cấu trúc Havard là 8031, PIC, Atmel AVR90S. Nếu sử dụng Chip 8031 chúng ta sẽ nhận thấy điều này thông qua việc truy nhập lấy dữ liệu ra từ vùng dữ liệu RAM hoặc từ vùng mã chương trình. Chúng ta có một vài con trỏ được sử dụng để lấy dữ liệu ra từ bộ nhớ dữ liệu RAM, nhưng chỉ có

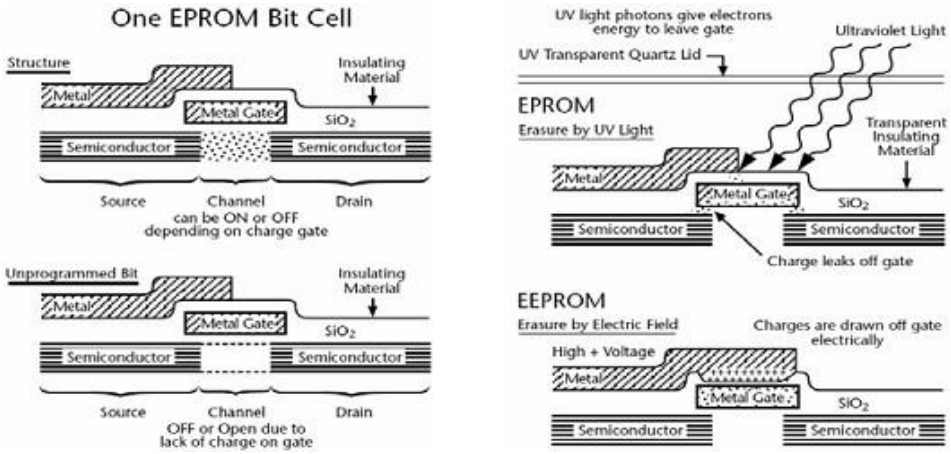
duy nhất một con trỏ DPTR có thể được sử dụng để lấy dữ liệu ra từ vùng mã chương trình. Hình 3.11 mô tả nguyên lý kiến trúc của bộ nhớ *von Neumann* và *Harvard*.

Ưu điểm nổi bật của cấu trúc bộ nhớ Harvard so với kiến trúc von Neumann là có hai kênh tách biệt để truy nhập vào vùng bộ nhớ mã chương trình và dữ liệu nhờ vậy mà mã chương trình và dữ liệu có thể được truy nhập đồng thời và làm tăng tốc độ luồng trao đổi với bộ xử lý.

Bộ nhớ chương trình – PROM (Programmable Read Only Memory)

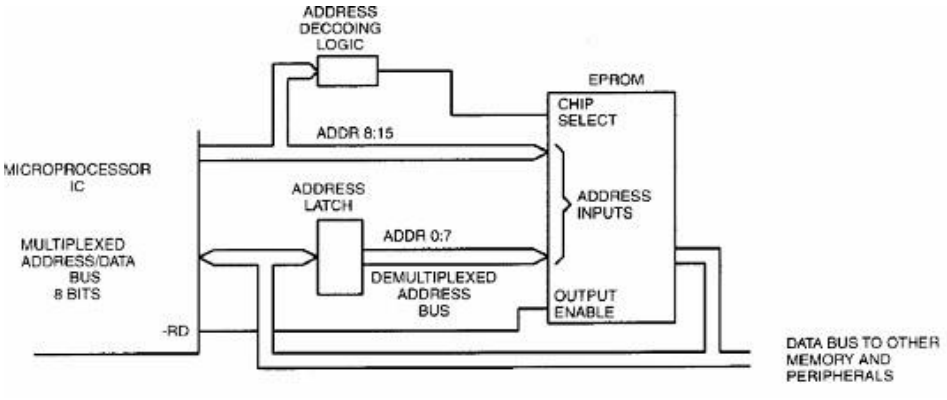
Vùng để lưu mã chương trình. Có ba loại bộ nhớ PROM thông dụng được sử dụng cho hệ nhúng và sẽ được giới thiệu lần lượt sau đây:

*EPROM :*Bao gồm một mảng các transistor khả trình. Mã chương trình sẽ được ghi trực tiếp và vi xử lý có thể đọc ra để thực hiện. EPROM có thể xoá được bằng tia cực tím và có thể được lập trình lại. Cấu trúc vật lý của EPROM được mô tả như trong Hình 3.12.



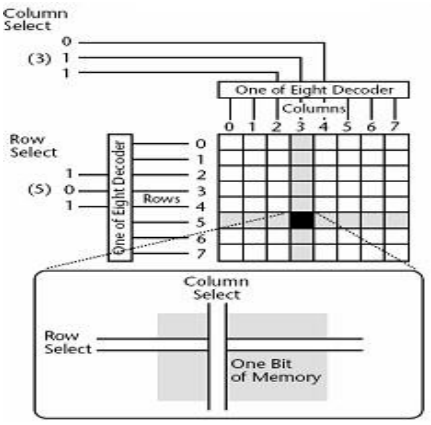
Nguyên lý cấu tạo và hoạt động xoá của EPROM

Bộ nhớ Flash: Cũng giống như EPROM được cấu tạo bởi một mảng transistor khả trình nhưng có thể xoá được bằng điện và chính vì vậy có thể nạp lại chương trình mà không cần tách ra khỏi nền phần cứng VXL. Ưu điểm của bộ nhớ flash là có thể lập trình trực tiếp trên mạch cứng mà nó đang thực thi trên đó.



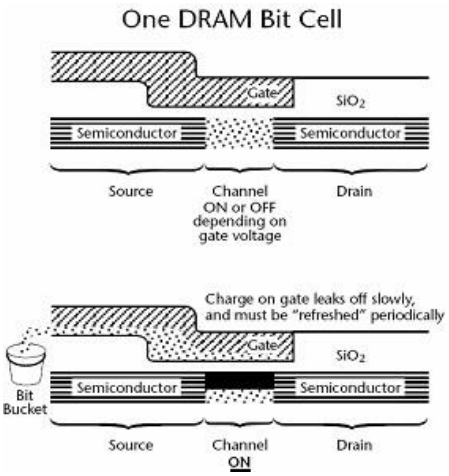
Sơ đồ nguyên lý ghép nối EPROM với VXL

Bộ nhớ dữ liệu – RAM: Vùng để lưu hoặc trao đổi dữ liệu trung gian trong quá trình thực hiện chương trình.

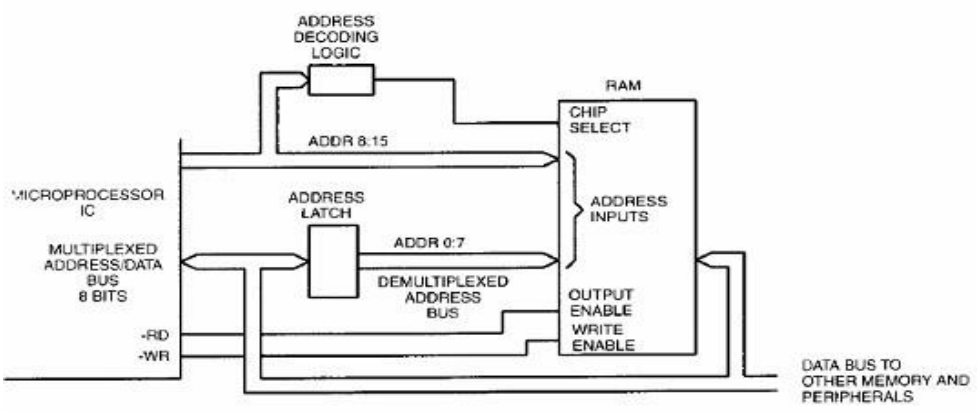


Cấu trúc nguyên lý bộ nhớ RAM

Có hai loại SRAM và DRAM.



Cấu trúc một phần tử nhớ DRAM

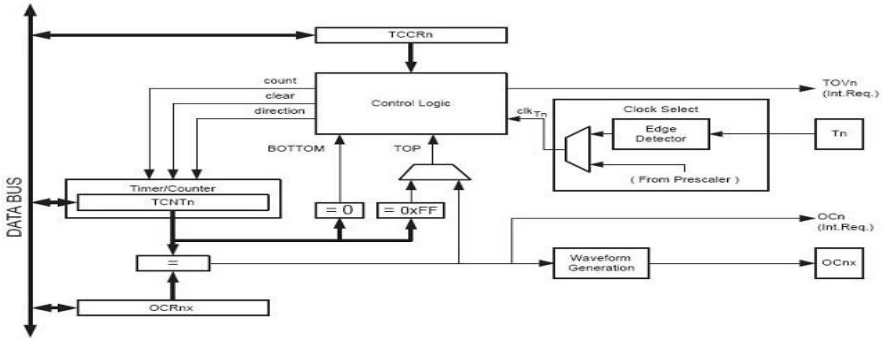


Nguyên lý ghép nối (mở rộng) RAM với VXL

Ngoại vi

Bộ định thời/Bộ đếm:

Hầu hết các chip vi điều khiển ngày nay đều có ít nhất một bộ định thời gian/bộ đếm có thể cấu hình hoạt động linh hoạt theo các mode phục vụ nhiều mục đích trong các ứng dụng xử lý, điều khiển. Các bộ định thời gian cho phép tạo ra các chuỗi xung và ngắt thời gian hoặc đếm theo các khoảng thời gian có thể lập trình. Chúng thường được ứng dụng phổ biến trong các nhiệm vụ đếm xung, đo khoảng thời gian các sự kiện, hoặc định chu kỳ thời gian thực thi các tác vụ. Một trong những ứng dụng quan trọng của bộ định thời gian là tạo nhịp từ bộ tạo xung thạch anh cho bộ truyền thông dị bộ đa năng hoạt động. Thực chất đó là ứng dụng để thực hiện phép chia tần số. Để đạt được độ chính xác, tần số thạch anh thường được chọn sao cho các phép chia số nguyên được thực hiện chính xác đảm bảo cho tốc độ truyền thông dữ liệu được tạo ra chính xác. Chính vì vậy họ vi điều khiển 80C51 thường hay sử dụng thạch anh có tần số dao động là 11.059 thay vì 12MHz để tạo ra nhịp hoạt động truyền thông tốc độ chuẩn 9600.



Bộ định thời/bộ đếm 8bit của AVR

Bộ điều khiển ngắt:

Ngắt là một sự kiện xảy ra làm dừng hoạt động chương trình hiện tại để phục vụ thực thi một tác vụ hay một chương trình khác. Cơ chế ngắt giúp CPU làm tăng tốc độ đáp ứng phục vụ các sự kiện trong chương trình hoạt động của VXL/VĐK. Các VĐK khác nhau sẽ định nghĩa các nguồn tạo ngắt khác nhau nhưng đều có chung một cơ chế hoạt động ví dụ như ngắt truyền thông nối tiếp, ngắt bộ định thời gian, ngắt cứng, ngắt ngoài...Khi một sự kiện yêu cầu ngắt xuất hiện, nếu được chấp nhận CPU sẽ lưu cất trạng thái hoạt động cho chương trình hiện tại đang thực hiện ví dụ như nội dung bộ đếm chương trình (con trỏ lệnh) các nội dung thanh ghi lưu dữ liệu điều khiển chương trình nói chung để thực thi chương trình phục vụ tác vụ cho sự kiện ngắt. Thực chất quá trình ngắt là CPU nhận dạng tín hiệu ngắt, nếu chấp nhận sẽ đưa con trỏ lệnh chương trình trỏ tới vùng mã chứa chương trình phục vụ tác vụ ngắt. Vì vậy mỗi một ngắt đều gắn với một vector ngắt như một con trỏ lưu thông tin địa chỉ của vùng bộ nhớ chứa mã chương trình phục vụ tác vụ của ngắt. CPU sẽ thực hiện chương trình phục vụ tác vụ ngắt đến khi nào gặp lệnh quay trở về chương trình trước thời điểm sự kiện ngắt xảy ra. Có thể phân ra 2 loại nguồn ngắt: *Ngắt cứng và Ngắt mềm*.

Ngắt mềm: Ngắt mềm thực chất thực hiện một lời gọi hàm đặc biệt mà được kích hoạt bởi các nguồn ngắt là các sự kiện xuất hiện từ bên trong chương trình và ngoại vi tích hợp trên Chip ví dụ như ngắt thời gian, ngắt chuyển đổi A/D, … Cơ chế ngắt này còn được hiểu là loại thực hiện đồng bộ với chương trình vì nó được kích hoạt và thực thi tại các thời điểm xác định trong chương trình. Hàm được gọi sẽ thực thi chức năng tương ứng với yêu cầu ngắt. Các hàm đó thường được trỏ bởi một vector ngắt mà đã được định nghĩa và gán cố định bởi nhà sản xuất Chip. Ví dụ như hệ điều hành của PC sử dụng ngắt số 21hex để gán cho ngắt truy nhập đọc dữ liệu từ đĩa cứng và xuất dữ liệu ra máy in.

Ngắt cứng: Ngắt cứng có thể được xem như là một lời gọi hàm đặc biệt trong đó nguồn kích hoạt là một sự kiện đến từ bên ngoài chương trình thông qua một cấu trúc phần cứng (thường được kết nối với thế giới bên ngoài qua các chân ngắt). Ngắt cứng thường được hiểu hoạt động theo cơ chế dị bộ vì các sự kiện ngắt kích hoạt từ các tín hiệu ngoại vi bên ngoài và tương đối độc lập với CPU, thường là không xác định được thời điểm kích hoạt. Khi các ngắt cứng được kích hoạt CPU sẽ nhận dạng và thực hiện lời gọi hàm thực thi chức năng phục vụ sự kiện ngắt tương ứng.

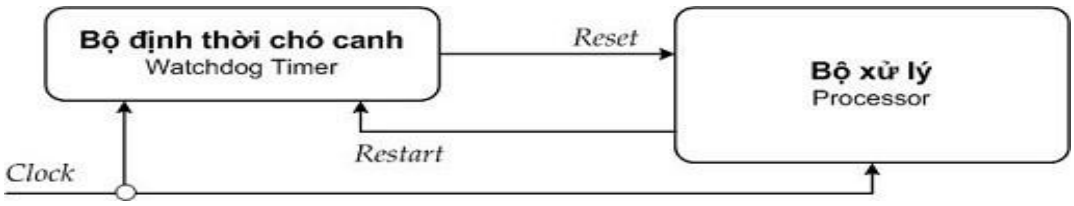
Trong các cơ chế ngắt khoảng thời gian từ khi xuất hiện sự kiện ngắt (có yêu cầu phục vụ ngắt) tới khi dịch vụ ngắt được thực thi là xác định và tuỳ thuộc vào công nghệ phần cứng xử lý của Chip.

Bộ định thời chó canh (Watchdog Timer)

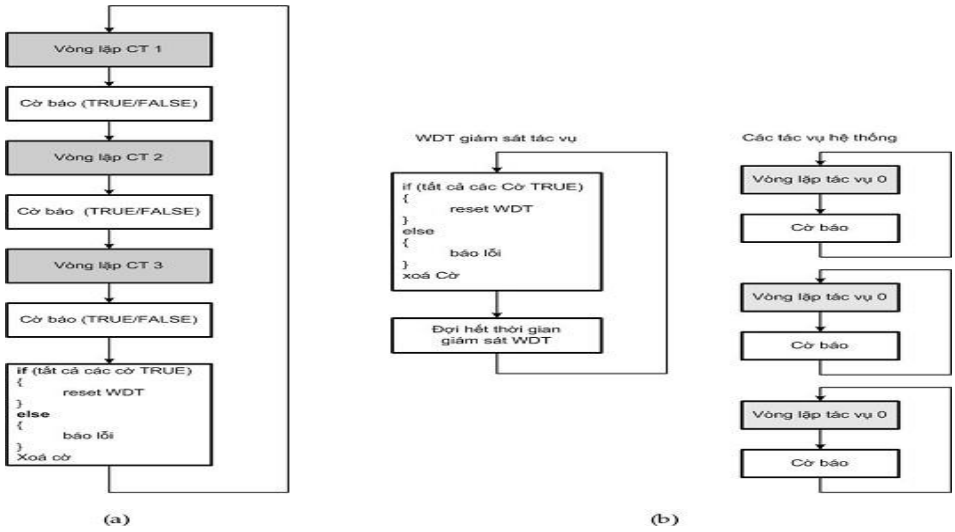
Thông thường khi có một sự cố xảy ra làm hệ thống bị treo hoặc chạy quẩn, CPU sẽ không thể tiếp tục thực hiện đúng chức năng. Đặc biệt khi hệ thống phải làm việc ở chế độ vận hành tự động và không có sự can thiệp trực tiếp thường xuyên bởi người

vận hành. Để thực hiện cơ chế tự giám sát và phát hiện sự cố phần mềm, một số VXL/ VĐK có thêm một bộ định thời chó canh. Bản chất đó là một bộ định thời đặc biệt để định nghĩa một khung thời gian hoạt động bình thường của hệ thống. Nếu có sự cố phần mềm xảy ra sẽ làm hệ thống bị treo khi đó bộ định thời chó canh sẽ phát hiện và giúp hệ thống thoát khỏi trạng thái đó bằng cách thực hiện khởi tạo lại chương trình. Chương trình hoạt động khi có bộ định thời phải đảm bảo reset nó trước khi khung thời gian bị vi phạm. Khung thời gian này được định nghĩa phụ thuộc vào sự đánh giá của người thực hiện phần mềm, thiết lập khoảng thời gian đảm bảo chắc chắn hệ thống thực hiện bình thường không có sự cố phần mềm.

Có một số cơ chế thực hiện cài đặt bộ định thời cho canh để giám sát hoạt động của hệ thống như sau:



Sơ đồ nguyên lý của bộ định thời

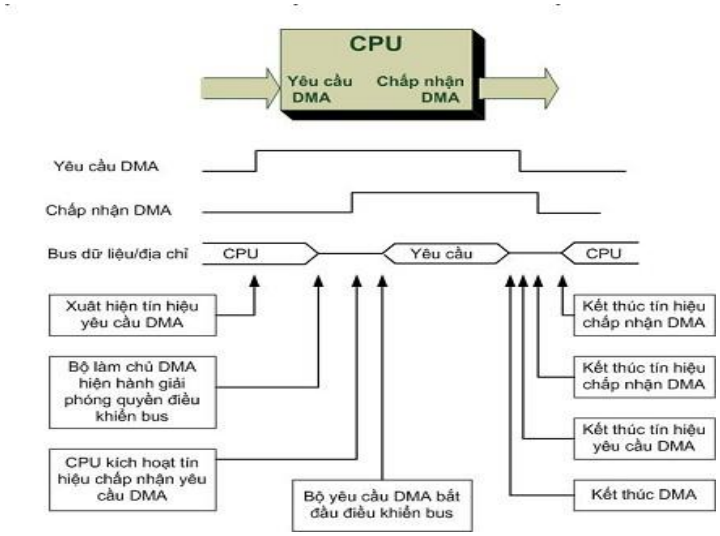


Nguyên lý hoạt động của bộ định thời

Bộ điều khiển truy cập bộ nhớ trực tiếp – DMA:

DMA (Direct Memory Access) là cơ chế hoạt động cho phép hai hay nhiều vi xử lý hoặc ngoại vi chia sẻ bus chung. Thiết bị nào đang có quyền điều khiển bus sẽ có thể toàn quyền truy nhập và trao đổi dữ liệu trực tiếp với các bộ nhớ như hệ thống có một vi xử lý. Ứng dụng phổ biến nhất của DMA là chia sẻ bộ nhớ chung giữa hai bộ vi xử lý hoặc các ngoại vi để truyền dữ liệu trực tiếp giữa thiết bị ngoại vi vào/ra và bộ nhớ dữ liệu của VXL.

Truy nhập bộ nhớ trực tiếp được sử dụng để đáp ứng nhu cầu trao đổi dữ liệu vào ra tốc độ cao giữa ngoại vi với bộ nhớ. Thông thường các ngoại vi kết nối với hệ thống phải chia sẻ bus dữ liệu và được điều khiển bởi CPU trong quá trình trao đổi dữ liệu. Điều này làm hạn chế tốc độ trao đổi, để tăng cường tốc độ và loại bỏ sự can thiệp của CPU, đặc biệt trong trường hợp cần truyền một lượng dữ liệu lớn. Cơ chế hoạt động DMA được mô tả như trong Hình 2.20. Thủ tục được bắt đầu bằng việc yêu cầu thực hiện DMA với CPU. Sau khi xử lý, nếu được chấp nhận CPU sẽ trao quyền điều khiển bus cho ngoại vi và thực hiện quá trình trao đổi dữ liệu. Sau khi thực hiện xong CPU sẽ nhận được thông báo và nhận lại quyền điều khiển bus. Trong cơ chế DMA, có hai cách để truyền dữ liệu: kiểu DMA chu kỳ đơn, và kiểu DMA chu kỳ nhóm (burst).



Nhịp hoạt động DMA

DMA chu kì đơn và nhóm: Trong kiểu hoạt động DMA chu kỳ nhóm, ngoại vi sẽ nhận được quyền điều khiển và truyền khối dữ liệu rồi trả lại quyền điều khiển cho CPU. Trong cơ chế DMA chu kỳ đơn ngoại vi sau khi nhân được quyền điều khiển bus chỉ truyền một từ dữ liệu rồi trả lại ngay quyền kiểm soát bộ nhớ và bus dữ liệu cho CPU. Trong cơ chế thực hiện DMA cần có một bước xử lý để quyết định xem thiết bị nào sẽ đươc nhận quyền điều khiển trong trường hợp có nhiều hơn một thiết bị có nhu cầu sử dụng DMA. Thông thường kiểu DMA chu kỳ nhóm cần ít dữ liệu thông tin điều khiển (overhead) nên có khả năng trao đổi với tốc độ cao nhưng lại chiếm nhiều thời gian truy nhập bus do truyền cả khối dữ liệu lớn. Điều này có thể ảnh hưởng đến hoạt động của cả hệ thống do trong suốt quá trình thực hiện DMA nhóm, CPU sẽ bị khoá quyền truy nhập bộ nhớ và không thể xử lý các nhiệm vụ khác của hệ thống mà có nhu cầu bộ nhớ, ví dụ như các dịch vụ ngắt, hoặc các tác vụ thời gian thực...

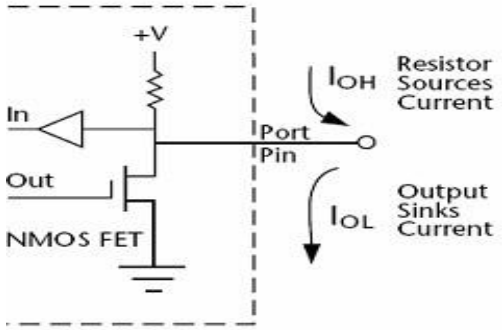
Chu kì rỗi (Cycle Stealing): Trong kiểu này DMA sẽ được thực hiện trong những thời điểm chu kỳ bus mà CPU không sử dụng bus do đó không cần thực hiện thủ tục xử lý cấp phát quyền truy nhập và thực hiện DMA. Hầu hết các vi xử lý hiện đại đều sử dụng gần như 100% dung lượng bộ nhớ và băng thông của bus nên sẽ không có nhiều thời

gian dành cho DMA thực hiện. Để tiết kiệm và tối ưu tài nguyên thì cần có một trọng tài phân xử và dữ liệu sẽ được truyền đi xếp chồng theo thời gian. Nói chung kiểu DMA dạng burst hiệu quả nhất khi khoảng thời gian cần thực hiện DMA tương đối nhỏ. Trong khoảng thời gian thực hiện DMA, toàn bộ băng thông của bus sẽ được sử dụng tối đa và toàn bộ khối dữ liệu sẽ được truyền đi trong một khoảng thời gian rất ngắn. Nhưng nhược điểm của nó là nếu dữ liệu cần truyền lớn và cần một khoảng thời gian dài thì sẽ dẫn đến việc block CPU và có thể bỏ qua việc xử lý các sự kiện và tác vụ khác. Đối với DMA chu kỳ đơn thì yêu cầu truy nhập bộ nhớ, truyền một từ dữ liệu và giải phóng bus. Cơ chế này cho phép thực hiện truyền interleave và được biết tới với tên gọi inteleaved DMA. Kiểu truyền DMA chu kỳ đơn phù hợp để truyền dữ liệu trong một khoảng thời gian dài mà có đủ thời gian để yêu cầu truy nhập và giải phóng bus cho mỗi lần truyền một từ dữ liệu. Chính vì vậy sẽ giảm băng thông truy nhập bus do phải mất nhiều thời gian để yêu cầu truy nhập và giải phóng bus. Trong trường hợp này CPU và các thiết bị khác vẫn có thể chia sẻ và truyền dữ liệu nhưng trong một dải băng thông hẹp. Trong nhiều hệ thống bus thực hiện cơ chế xử lý và giải quyết yêu cầu truy nhập (trọng tài) thông qua dữ liệu truyền vì vậy cũng không ảnh hưởng nhiều đến tốc độ truyền DMA. DMA được yêu cầu khi khả năng điều khiển của CPU để truyền dữ liệu thực hiện quá chậm. DMA cũng thực sự có ý nghĩa khi CPU đang phải thực hiện các tác vụ khác mà không cần nhu cầu truy nhập bus.

Giao diện

Giao diện song song 8bit/16bit

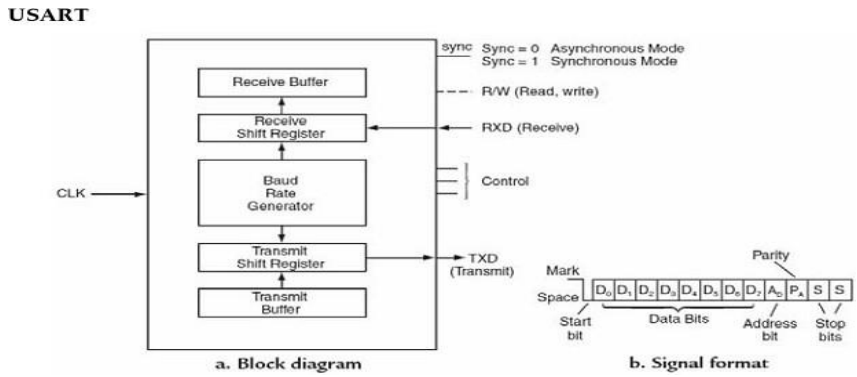
Các cổng song song là một dạng giao diện vào ra đơn giản và phổ biến nhất để kết nối thông tin với ngoại vi. Có nhiều loại cấu trúc giao diện vật lý điện tử từ dạng cổng vào ra đơn giản cực Collector TTL hở trong các ứng dụng cổng máy in đến các loại cấu trúc giao diện cổng tốc độ cao như các chuẩn bus IEEE-488 hay SCSI. Hầu hết các chip điều khiển nhúng có một vài cổng vào ra song song khả trình (có thể cấu hình). Các giao diện đó phù hợp với các cổng vào ra đơn giản như các khoá chuyển. Chúng cũng phù hợp trong các bài toán phục vụ giao diện kết nối điều khiển và giám sát theo các giao diện như kiểu rơle bán dẫn.



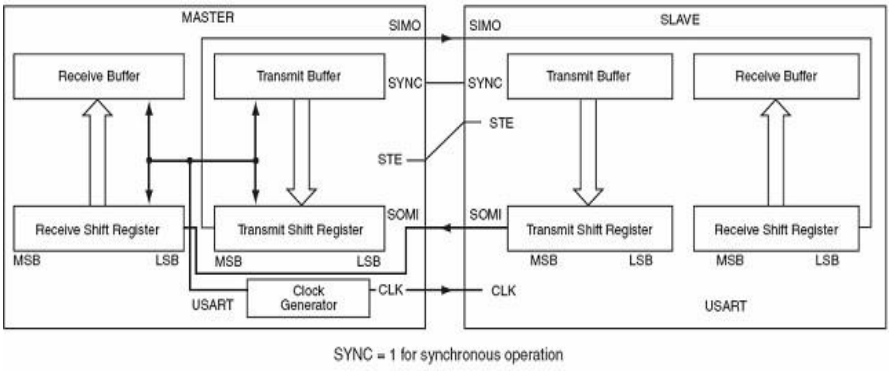
Cấu trúc nguyên lý điển hình của một cổng vào/ra logic

Giao diện nối tiếp:

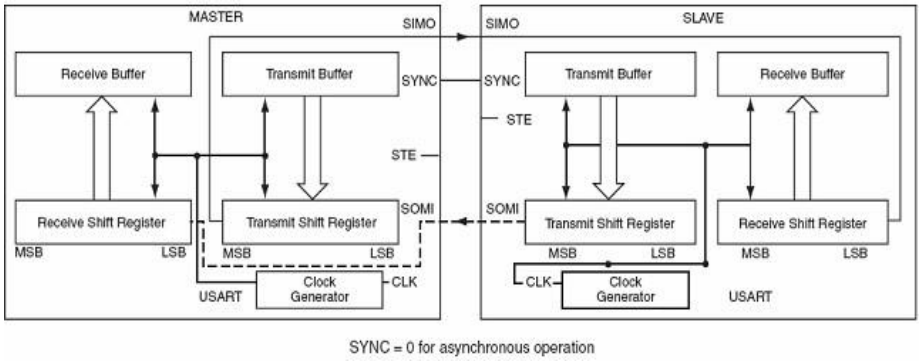
USART:



Cấu trúc đơn giản hóa cuae USART



Mode hoạt động truyền thông đồng bộ



Mode hoạt động truyền thông dị bộ

I2C (Inter-IC)

Giao thức ưu tiên truyền thông nối tiếp được phát triển bởi Philips Semiconductor và được gọi là bus I2C. Vì nguồn gốc nó được thiết kế là để điều khiển liên thông IC (Inter IC) nên nó được đặt tên là I2C. Tất cả các chíp có tích hợp và tương thích với I2C đều có thêm một giao diện tích hợp trên Chip để truyền thông trực tiếp với các thiết bị tương

thích I2C khác. Việc truyền dữ liệu nối tiếp theo hai hướng 8 bit được thực thi theo 3 chế độ sau:

- Chuẩn (Standard)—100 Kbits/sec
- Nhanh (Fast)—400 Kbits/sec
- Tốc độ cao (High Speed)—3.4 Mbits/sec

Đường bus thực hiện truyền thông nối tiếp I2C gồm hai đường là đường truyền dữ liệu nối tiếp SDA và đường truyền nhịp xung đồng hồ nối tiếp SCL. Vì cơ chế hoạt động là đồng bộ nên nó cần có một nhịp xung tín hiệu đồng bộ. Các thiết bị hỗ trợ I2C đều có một địa chỉ định nghĩa trước, trong đó một số bit địa chỉ là thấp có thể cấu hình. Đơn vị hoặc thiết bị khởi tạo quá trình truyền thông là đơn vị Chủ và cũng là đơn vị tạo xung nhịp đồng bộ, điều khiển cho phép kết thúc quá trình truyền. Nếu đơn vị Chủ muốn truyền thông với đơn vị khác nó sẽ gửi kèm thông tin địa chỉ của đơn vị mà nó muốn truyền trong dữ liệu truyền. Đơn vị Tớ đều được gán và đánh địa chỉ thông qua đó đơn vị Chủ có thể thiết lập truyền thông và trao đổi dữ liệu. Bus dữ liệu được thiết kế để cho phép thực hiện nhiều đơn vị Chủ và Tớ ở trên cùng Bus.

Quá trình truyền thông I2C được bắt đầu bằng tín hiệu start tạo ra bởi đơn vị Chủ. Sau đó đơn vị Chủ sẽ truyền đi dữ liệu 7 bit chứa địa chỉ của đơn vị Tớ mà nó muốn truyền thông, theo thứ tự là các bit có trọng số lớn nhất MSB sẽ được truyền trước. Bit thứ tám tiếp theo sẽ chứa thông tin để xác định đơn vị Tớ sẽ thực hiện vai trò nhận (0) hay gửi (1) dữ liệu. Tiếp theo sẽ là một bit ACK xác nhận bởi đơn vị nhận đã nhận được 1 byte trước đó hay không. Đơn vị truyền (gửi) sẽ truyền đi 1 byte dữ liệu bắt đầu bởi MSB. Tại điểm cuối của byte truyền, đơn vị nhận sẽ tạo ra một bit xác nhận ACK mới. Khuôn mẫu 9 bit này (gồm 8 bit dữ liệu và 1 bit xác nhận) sẽ được lặp lại nếu cần truyền tiếp byte nữa. Khi đơn vị Chủ đã trao đổi xong dữ liệu cần nó sẽ quan sát bit xác nhận ACK cuối cùng rồi sau đó sẽ tạo ra một tín hiệu dừng STOP để kết thúc quá trình truyền thông.

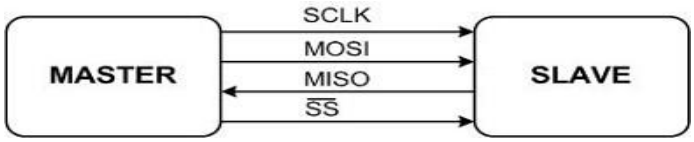
I2C là một giao diện truyền thông đặc biệt thích hợp cho các ứng dụng truyền thông giữa các đơn vị trên cùng một bo mạch với khoảng cách ngắn và tốc độ thấp. Ví dụ như truyền thông giữa CPU với các khối chức năng trên cùng một bo mạch như EEPROM, cảm biến, đồng hồ tạo thời gian thực... Hầu hết các thiết bị hỗ trợ I2C hoạt động ở tốc độ 400Kbps, một số cho phép hoạt động ở tốc độ cao vài Mbps. I2C khá đơn giản để thực thi kết nối nhiều đơn vị vì nó hỗ trợ cơ chế xác định địa chỉ.

SPI:

SPI là một giao diện cổng nối tiếp đồng bộ ba dây cho phép kết nối truyền thông nhiều VĐK được phát triển bởi Motorola. Trong cấu hình mạng kết nối truyền thống này phải có một VĐK giữ vai trò là Chủ (Master) và các VĐK còn lại có thể hoặc là Chủ hoặc là Tớ. SPI có 4 tốc độ có thể lập trình, cực và pha nhịp đồng hồ khả trình và kết thúc ngắt

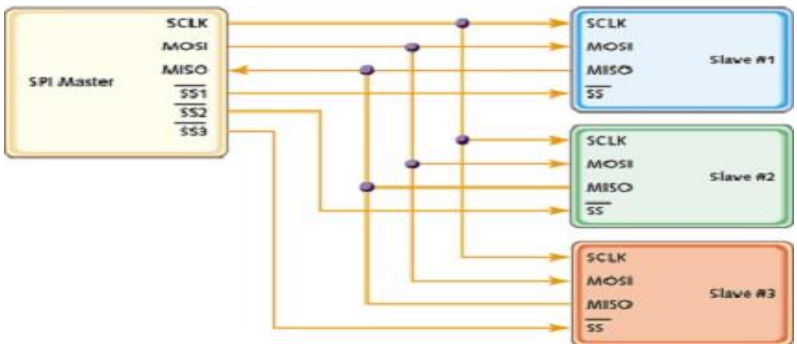
truyền thông. Nhịp đồng hồ không nằm trong dòng dữ liệu và phải được cung cấp như một tín hiệu tách độc lập. Có ba thanh ghi SPSR, SPCR và SPDR cho phép thực hiện các chức năng điều khiển, trạng thái và lữu trữ. Có bốn chân cơ bản cần thiết để thực thi chuẩn giao diện truyền thông này:

- Dữ liệu ra MOSI (Master Output – Slave Input)
- Dữ liệu vào MISO (Master Input – Slave Output)
- Nhịp xung chuẩn SCLK (Serial Clock)
- Lựa chọn thành phần tớ SS (Slave Select)



Kết nối nguyên lý truyền thông SPI giữa mộy Master và Slave

Hình 2.25 chỉ ra nguyên lý kết nối giữa một đơn vị Chủ và một đơn vị Tớ trong truyền thông SPI. Trong đó tín hiệu SCLK sẽ được tạo ra bởi đơn vị Chủ và là tín hiệu vào của đơn vị Tớ. MOSI là đường truyền dữ liệu ra từ đơn vị Chủ tới đơn vị Tớ và MISO là đường truyền dữ liệu vào đơn vị Chủ đến từ đơn vị Tớ. Đơn vị Tớ được lựa chọn khi đơn vị Chủ kích hoạt tín hiệu SS.



Sơ đồ kết nối truyền thống SPI của một đơn vị chủ với nhiều đơn vị tớ

Nếu hệ thống có nhiều đơn vị tớ đơn vị Chủ sẽ tạo phải ra các tín hiệu tách biệt để chọn đơn vị Tớ. Cơ chế đó được thực hiện nhờ sơ đồ kết nối nguyên lý mô tả như trong Hình 2.26. Đơn vị Chủ sẽ tạo ra tín hiệu chọn đơn vị Tớ nhờ các chân tín hiệu logic đa chức năng. Các tín hiệu này phải được điều khiển và đảm bảo ổn định về thời gian để tránh trường hợp tín hiệu bị thay đổi trong quá trình đang truyền dữ liệu. Một điều dễ nhận ra rằng SPI không hỗ trợ cơ chế xác nhận trong quá trình thực hiện truyền thông. Điều này phụ thuộc vào giao thức định nghĩa hoặc phải thực hiện bổ sung thêm một số các mở rộng phụ bên ngoài.

Khả năng truyền thông đồng thời hai chiều với tốc độ lên đến khoảng vài Mbit/s và nguyên lý khá đơn giản nên SPI hoàn toàn phù hợp để thực hiện truyền thông giữa các

thiết bị yêu cầu truyền thông tốc độ chậm, đặc biệt hiệu quả trong các ứng dụng một đơn vị Chủ và một đơn vị Tớ. Tuy nhiên trong các ứng dụng với nhiều đơn vị Tớ việc thực thi lại khá phức tạp vì thiếu cơ chế xác định địa chỉ, và sự phức tạp sẽ tăng lên khi số đơn vị Tớ tăng.

Một số nền phần cứng và Cơ sở kỹ thuật của phần mềm nhúng

Một số nền phần cứng nhúng thông dụng

Trong phần này giới thiệu ngắn gọn cấu trúc nguyên lý của các chip xử lý nhúng ứng dụng trong các nền phần cứng nhúng hiện nay.

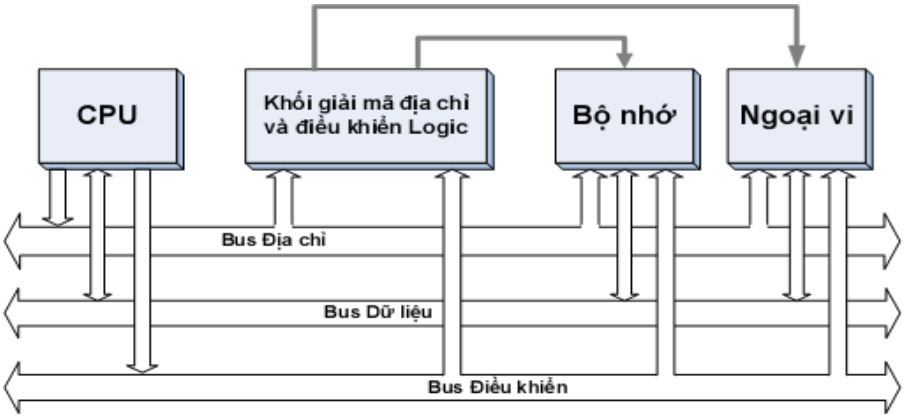
Sự phát triển nhanh chóng các chủng loại Chip khả trình với mật độ tích hợp cao đã và đang có một tác động đáng kể đến sự thay đổi trong việc thiết kế các nền phần cứng thiết bị xử lý và điều khiển số trong thập kỷ gần đây. Mỗi chủng loại đều có những đặc điểm và phạm vi đối tượng ứng dụng và luôn không ngừng phát triển để đáp ứng một cách tốt nhất cho các yêu cầu công nghệ. Chúng đang hướng tới tập trung cho một thị trường công nghệ tiềm năng rộng lớn đó là các thiết bị xử lý và điều khiển nhúng. Ở đây giới thiệu ngắn gọn về 2 chủng loại chip xử lý, điều khiển nhúng điển hình đang tồn tại và phát triển về một số đặc điểm và hướng phạm vi ứng dụng của chúng.

Có thể kể ra hàng loạt các Chíp khả trình có thể sử dụng cho các bài toán thiết kế hệ nhúng như các họ vi xử lý/vi điều khiển nhúng (Microprocessor/ Microcontroller), Chip DSP (Digital Signal Processing), các Chip khả trình trường (FPD – Field Programmable Device). Chúng ta dễ bị choáng ngợp nếu bắt đầu công việc thiết kế bằng việc tìm kiếm một Chip xử lý điều khiển phù hợp cho ứng dụng. Vì vậy cần phải có một hiểu biết và sự phân biệt về đặc điểm và ứng dụng của chúng khi lựa chọn và thiết kế. Các thông tin liên quan như nhà sản xuất cung cấp Chip, các kiến thức và công cụ phát triển kèm theo…Một số chủng loại Chip điển hình sẽ được giới thiệu.

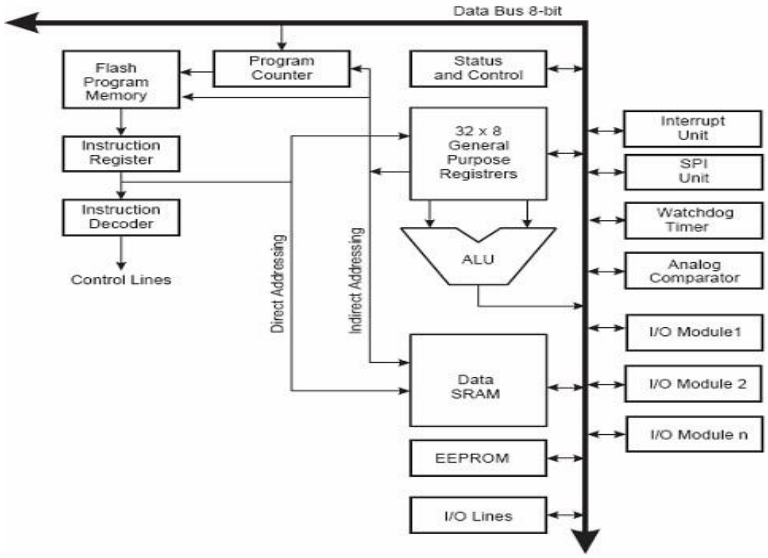
Chip Vi xử lý/Vi điều khiển nhúng

Đây là một chủng loại rất điển hình và đang được sử dụng rất phổ biến hiện này. Chúng được ra đời và sử dụng theo sự phát triển của các Chip xử lý ứng dụng cho máy tính. Vì đối tượng ứng dụng là các thiết bị nhúng nên cấu trúc cũng được thay đổi theo để đáp ứng các ứng dụng. Hiện nay chúng ta có thể thấy các họ vi xử lý điều khiển của rất nhiều các nhà chế tạo cung cấp như, Intel, Atmel, Motorola, Infineon. Về cấu trúc, chúng cũng tương tự như các Chíp xử lý phát triển cho PC nhưng ở mức độ đơn giản hơn nhiều về công năng và tài nguyên. Phổ biến vẫn là các Chip có độ rộng bus dữ liệu là 8bit, 16bit, 32bit. Về bản chất cấu trúc, Chip vi điều khiển là chip vi xử lý được tích hợp thêm các ngoại vi. Các ngoại vi thường là các khối chức năng ngoại vi thông dụng như bộ định thời gian, bộ đếm, bộ chuyển đổi A/D, giao diện song song, nối tiếp…Mức độ tích hợp ngoại vi cũng khác nhau tuỳ thuộc vào mục đích ứng dụng sẽ có thể tìm được Chip phù hợp. Thực tế với các ứng dụng yêu cầu độ tích hợp cao thì sẽ sử dụng giải pháp tích hợp

trên chip, nếu không thì hầu hết các Chip đều cung cấp giải pháp để mở rộng ngoại vi đáp ứng cho một số lượng ứng dụng rộng và mềm dẻo.



Kiến trúc nguyên lý của VĐK với cấu trúc Havard



Kiến trúc của họ VĐK AVR

Chip DSP

DSP vẫn được biết tới như một loại vi điều khiển đặc biệt với khả năng xử lý nhanh để phục vụ các bài toán yêu cầu khối lượng và tốc độ xử lý tính toán lớn. Với ưu điểm nổi bật về độ rộng băng thông của bus và thanh ghi tích luỹ, cho phép ALU xử lý song song với tốc độ đọc và xử lý lệnh nhanh hơn các loại vi điều khiển thông thường. Chip DSP cho phép thực hiện nhiều lệnh trong một nhịp nhờ vào kiến trúc bộ nhớ *Havard*.

Thông thường khi phải sử dụng DSP tức là để đáp ứng các bài toán tính toán lớn và tốc độ cao vì vậy định dạng biểu diễn toán học sẽ là một yếu tố quan trọng để phân loại và được quan tâm. Hiện nay chủ yếu chúng vẫn được phân loại theo hai kiểu là dấu phảy động và dấu phảy tĩnh. Đây cũng chính là một yếu tố quan trọng phải quan tâm đối với người thiết kế để lựa chọn được một DSP phù hợp với ứng dụng của mình. Các loại

DSP dấu phảy tĩnh thường là loại 16bit hoặc 24bit còn các loại dấu phảy tĩnh thường là 32bit. Một ví dụ điển hình về một DSP 16bit dấu phảy tĩnh là TMS320C55x, lưu các số nguyên 16bit hoặc các số thực trong một miền giá trị cố định. Tuy nhiên các giá trị và hệ số trung gian có thể được lưu trữ với độ chính xác là 32bit trong thanh ghi tích luỹ 40bit nhằm giảm thiểu lỗi tính toán do phép làm tròn trong quá trính tính toán. Thông thường các loại DSP dấu phảy tĩnh có giá thành rẻ hơn các loại DSP dấu phảy động vì yêu cầu số lượng chân Onchip ít hơn và cần sử dụng lượng *silicon* ít hơn.

Ưu điểm nổi bật của các DSP dấu phảy động là có thể xử lý và biểu diễn số trong dải phạm vi giá trị rộng và động. Do đó vấn đề về chuyển đổi và hạn chế về phạm vi biểu diễn số không phải quan tâm như đối với loại DSP dấu phảy tĩnh. Một loại DSP 32bit dấu phảy tĩnh điển hình là TMS320C67x có thể xử lý và biểu diễn số gồm 24bit *mantissa* và 8bit *exponent*. Phần *mantissa* biểu diễn phần số lẻ trong phạm vi -1.0 → +1.0 và phần *exponent* biểu diễn vị trí của dấu phảy nhị phân và có thể dịch chuyển sang trái hoặc phải tuỳ theo giá trị số mà nó biểu diễn. Điều này trái ngược với các thiết kế trên nền DSP dấu phảy tĩnh, người phát triển chương trình phải tự qui ước, tính toán và phân chia ấn định thang biểu diễn số và phải luôn lưu tâm tới khả năng tràn số có thể xảy ra trong quá trình xử lý tính toán. Chính điều này đã gây ra khó khăn không nhỏ đối với người lập trình. Nói chung phát triển chương trình cho DSP dấu phảy động thường đơn giản hơn nhưng giá thành lại cao hơn nhiều và năng lượng tiêu thụ thông thường cũng lớn hơn.

Ví dụ độ chính xác của DSP dấu phảy động 32 bit là 2−23 với 24 bit biểu diễn phần *mantissa*. Vùng động là ($1.18 \times 10^{-38} \leq x \leq 3.4 \times 10^{38}$).

Những nhà thiết kế hệ thống phải quyết định vùng và độ chính xác cần thiết cho các ứng dụng. Các vi xử lý dấu phảy động thường được sử dụng cho các ứng dụng yêu cầu về độ chính xác cao và dải biểu diễn số lớn phù hợp với hệ thống có cấu trúc bộ nhớ lớn. Hơn nữa các DSP dấu phảy động cho phép phát triển phần mềm hiệu quả và đơn giản hơn bằng các trình biên dịch ngôn ngữ bậc cao như C do đó có thể giảm được giá thành và thời gian phát triển. Tuy nhiên giá thành lại cao nên các DSP dấu phảy động phù hợp với các ứng dụng khá đặc biệt và thường là với số lượng ít.

Cơ sở kỹ thuật của phần mềm nhúng

Phần mềm nhúng là gì?

Phần mềm nhúng là một chương trình được viết, biên dịch trên máy tính và nạp vào một hệ thống khác (gọi tắt là KIT) bao gồm một hoặc nhiều bộ vi xử lý đã được cài sẵn một hệ điều hành, bộ nhớ ghi chép được, các cổng giao tiếp với các phần cứng khác…

Phần mềm nhúng là phần mềm tạo nên phần hồn, phần trí tuệ của các sản phẩm nhúng. Phần mềm nhúng ngày càng có tỷ lệ giá trị cao trong giá trị của các sản phẩm nhúng.

Hiện nay phần lớn các phần mềm nhúng nằm trong các sản phẩm truyền thông và các sản phẩm điện tử tiêu dùng (consumer electronics), tiếp đến là trong các sản phẩm ô tô, phương tiện vận chuyển, máy móc thiết bị y tế, các thiết bị năng lượng, các thiết bị cảnh báo bảo vệ và các sản phẩm đo và điều khiển.

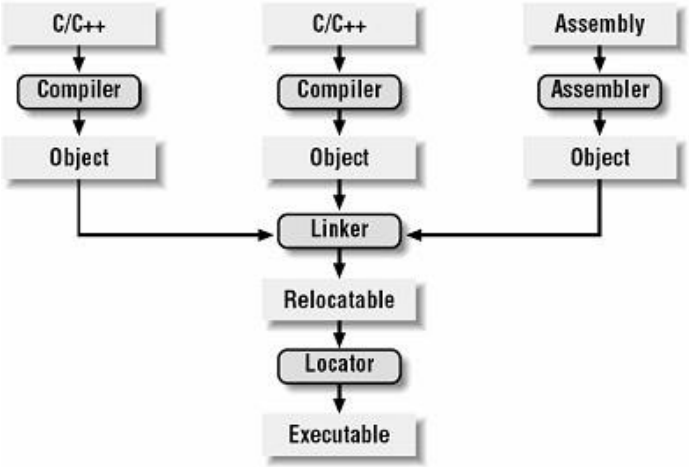
Để có thể tồn tại và phát triển, các sản phẩm công nghiệp và tiêu dùng cần phải thường xuyên đổi mới và ngày càng có nhiều chức năng tiện dụng và thông minh hơn. Các chức năng này phần lớn do các chương trình nhúng tạo nên. Phần mềm nhúng là một lĩnh vực công nghệ then chốt cho sự phát triển kinh tế của nhiều quốc gia trên thế giới như Nhật Bản, Hàn Quốc, Phần Lan và Trung quốc. Tại Mỹ có nhiều chương trình hỗ trợ của Nhà nước để phát triển các hệ thống và phần mềm nhúng. Hàn Quốc có những dự án lớn nhằm phát triển công nghệ phần mềm nhúng như các thiết bị gia dụng nối mạng Internet, hệ thống phần mềm nhúng cho phát triển thành phố thông minh, dự án phát triển ngành công nghiệp phần mềm nhúng, trung tâm hỗ trợ các ngành công nghiệp hậu PC. Thụy Điển coi phát triển các hệ nhúng có tầm quan trọng chiến lược cho sự phát triển của đất nước. Phần Lan có những chính sách quốc gia tích cực cho nghiên cứu phát triển các hệ nhúng đặc biệt là các phần mềm nhúng. Những quốc gia này còn thành lập nhiều viện nghiên cứu và trung tâm phát triển các hệ nhúng.

Đặc điểm của phần mềm nhúng

Hiện nay phần mềm nhúng có một số đặc điểm sau nổi bật:

- Phần mềm nhúng phát triển theo hướng chức năng hóa đặc thù.
- Hạn chế về tài nguyên bộ nhớ.
- Yêu cầu thời gian thực.

Quy trình phát triển của phần mềm nhúng



Quá trình biên dịch và phát triển phần mềm nhúng

- Quá trình biên dịch (Computing):

Nhiệm vụ chính của bộ biên dịch là chuyển đổi chương trình được viết bằng ngôn ngữ thân thiện với con người ví dụ như C, C++,…thành tập mã lệnh tương đương có thể đọc và hiểu bởi bộ vi xử lý đích. Theo cách hiểu này thì bản chất một bộ hợp ngữ cũng là một bộ biên dịch để chuyển đổi một - một từ một dòng lệnh hợp ngữ thành một dạng mã lệnh tương đương cho bộ vi xử lý có thể hiểu và thực thi. Chính vì vậy đôi khi người ta vẫn nhầm hiểu giữa khái niệm bộ hợp ngữ và bộ biên dịch. Tuy nhiên việc biên dịch của bộ hợp ngữ sẽ được thực thi đơn giản hơn rất nhiều so với các bộ biên dịch cho các mã nguồn viết bằng ngôn ngữ bậc cao khác.

Mỗi một bộ xử lý thường có riêng ngôn ngữ máy vì vậy cần phải chọn lựa một bộ biên dịch phù hợp để có thể chuyển đổi chính xác thành dạng mã máy tương ứng với bộ xử lý đích. Đối với các hệ thống nhúng, bộ biên dịch là một chương trình ứng dụng luôn được thực thi trên máy chủ (môi trường phát triển chương trình) và còn có tên gọi là bộ biên dịch chéo (cross - compiler). Vì bộ biên dịch chạy trên một nền phần cứng để tạo ra mã chương trình chạy trên môi trường phần cứng khác. Việc sử dụng bộ biên dịch chéo này là một thành phần không thể thiếu trong quá trình phát triển phần mềm cho hệ nhúng. Các bộ biên dịch chéo thường có thể cấu hình để thực thi việc chuyển đổi cho nhiều nền phần cứng khác nhau một cách linh hoạt. Và việc lựa chọn cấu hình biên dịch tương ứng với các nền phần cứng đôi khi cũng khá độc lập với chương trình ứng dụng của bộ biên dịch.

Kết quả đầu tiên của quá trình biên dịch nhận được là một dạng mã lệnh được biết tới với tên gọi là tệp đối tượng (object file). Nội dung của tệp đối tượng này có thể được xem như là một cấu trúc dữ liệu trung gian và thường được định nghĩa như một định dạng chuẩn COFF (Common Object File Format) hay định dạng của bộ liên kết mở rộng ELF (Extended Linker Format)… Nếu sử dụng nhiều bộ biên dịch cho các modul mã nguồn của một chương trình lớn thì cần phải đảm bảo rằng các tệp đối tượng được tạo ra phải có chung một kiểu định dạng.

Hầu hết nội dung của các tệp đối tượng đều bắt đầu bởi một phần header để mô tả các phần theo sau. Mỗi một phần sẽ chứa một hoặc nhiều khối mã hoặc dữ liệu như được sử dụng trong tệp mã nguồn. Tuy nhiên các khối đó được nhóm lại bởi bộ biên dịch vào trong các phần liên quan. Ví dụ như tất cả các khối mã được nhóm lại vào trong một phần được gọi là text, các biến toàn cục đã được khởi tạo (cùng các giá trị khởi tạo của chúng) vào trong phần dữ liệu, và các biến toàn cục chưa được khởi tạo vào trong phần *bss*.

Cũng khá phổ biến thường có một bảng biểu tượng chứa trong nội dung của tệp đối tượng. Nó chứa tên và địa chỉ của tất cả các biến và hàm được tham chiếu trong tệp mã nguồn. Các phần chứa trong bảng này không phải lúc nào cũng đầy đủ vì có một số biến và hàm được định nghĩa và chứa trong các tệp mã nguồn khác. Chính vì vậy cần phải có bộ liên kết để thực thi xử lý vấn đề này.

- Quá trình liên kết (Linking):

Tất cả các tệp đối tượng nhận được sau bước thực hiện biên dịch đầu tiên đều phải được tổ hợp lại theo một cách đặc biệt trước khi nó được nạp và chạy ở trên môi trường phần cứng đích. Như đã thấy ở trên, bản thân các tệp đối tượng cũng có thể là chưa hoàn thiện vì vậy bộ liên kết phải xử lý để tổ hợp các tệp đối tượng đó với nhau và hoàn thiện nốt phần còn khuyết cho các biến hoặc hàm tham chiếu liên thông giữa các tệp mã nguồn được biên dịch độc lập.

Kết quả đầu ra của bộ liên kết là một tệp đối tượng mới có chứa tất cả mã và dữ liệu trong tệp mã nguồn và cùng kiểu định dạng tệp. Nó thực thi được điều này bằng cách tổ hợp một cách tương ứng các phần text, dữ liệu và phần bss …từ các tệp đầu vào và tạo ra một tệp đối tượng theo định dạng mã máy thống nhất. Trong qúa trình bộ liên kết thực hiện tổ hợp các phần nội dung tương ứng nó còn thực hiện thêm cả vấn đề hoàn chỉnh các địa chỉ tham chiếu của các biến và hàm chưa được đầy đủ trong bước thực hiện biên dịch.

Các bộ liên kết có thể được kích hoạt thực hiện độc lập với bộ biên dịch và các tệp đối tượng được tạo ra bởi bộ biên dịch được coi như các tham biến vào. Đối với các ứng dụng nhúng nó thường chứa phần mã khởi tạo đã được biên dịch cũng phải được gộp ở trong danh sách tham biến vào này.

Nếu cùng một biểu tượng được khai báo hơn một lần nằm trong một tệp đối tượng thì bộ liên kết sẽ không thể xử lý. Nó sẽ kích hoạt cơ chế báo lỗi để người phát triển chương trình xem xét lại. Hoặc khi một biểu tượng không thể tìm được địa chỉ tham chiếu thực trong toàn bộ các tệp đối tượng thì bộ liên kết sẽ cố gắng tự giải quyết theo khả năng cho phép dựa vào các thông tin ví dụ như chứa trong phần mô tả của thư viện chuẩn. Điều này cũng thường hoặc có thể gặp với trường hợp các hàm tham chiếu trong chương trình.

Rất đáng tiếc là các hàm thư viện chuẩn thường yêu cầu một vài thay đổi trước khi nó có thể được sử dụng trong chương trình ứng dụng nhúng. Vấn đề ở đây là các thư viện chuẩn cung cấp cho các bộ công cụ phát triển chỉ dừng đến khả năng định dạng và tạo ra tệp đối tượng. Hơn nữa chúng ta cũng rất ít khi có thể truy nhập được vào mã nguồn của các thư viện chuẩn để có thể tự thay đổi. Hiện nay cũng có một số nhà cung cấp dịch vụ phần mềm hỗ trợ công cụ chuyển đổi hay thay đổi thư viện C chuẩn để ứng dụng cho các ứng dụng nhúng, ví dụ như *Cygnus*. Gói phần mềm này được gọi là *newlib* và được cung cấp miễn phí. Chúng ta có thể tải về trang web của Cygnus. Nó sẽ hỗ trợ chúng ta giải quyết vấn đề mà bộ liên kết có thể gặp phải khi chương trình sử dụng các hàm thuộc thư viện C chuẩn.

Sau khi đã hợp nhất thành công tất cả các thành phần mã và phần dữ liệu tương ứng cũng như các vấn đề về tham chiếu tới các biểu tượng chưa được thực thi trong quá trình

biên dịch đơn lẻ, bộ liên kết sẽ tạo ra một bản sao đặc biệt của chương trình có khả năng định vị lại (relocatable). Hay nói cách khác, chương trình được hoàn thiện ngoại trừ một điều: Không có địa chỉ bộ nhớ nào chưa được gán bên trong các phần mã và dữ liệu. Nếu chúng ta không phải là đang phát triển phần mềm cho hệ nhúng thì quá trình biên dịch có thể kết thúc tại đây. Tuy nhiên, với hệ nhúng ngay cả hệ thống nhúng đã bao gồm cả hệ điều hành chúng ta vẫn cần phải có một mã chương trình (image) nhị phân được định vị tuyệt đối. Thực tế nếu có một hệ điều hành thì phần mã và dữ liệu cũng thường gộp cả vào bên trong chương trình có khả năng định vị lại. Toàn bộ ứng dụng nhúng bao gồm cả hệ điều hành thường liên kết tĩnh với nhau và thực hiện như một mã chương trình nhị phân thống nhất.

- Quá trình định vị (Locating)

Công cụ thực hiện việc chuyển đổi một chương trình có khả năng định vị lại thành một dạng mã chương trình nhị phân có thể thực thi được gọi là bộ định vị. Nó sẽ đảm nhiệm vai trò của bước đơn giản nhất trong các bước thực thi biên dịch nói chung. Thực tế chúng ta phải tự làm hầu hết công việc của bước này bằng cách cung cấp thông tin về bộ nhớ đã được cấu hình trên nền phần cứng mà chúng ta đang phát triển và đó chính là tham số đầu vào cho việc thực thi của bộ định vị. Bộ định vị sẽ sử dụng thông tin này để gán các địa chỉ vật lý cho mỗi phần mã lệnh và dữ liệu bên trong chương trình được thực thi mà có thể định vị lại. Tiếp theo nó sẽ tạo ra một tệp có chứa chương trình bộ nhớ nhị phân để có thể nạp trực tiếp vào bộ nhớ chương trình trên nền phần cứng thực thi.

Trong nhiều trường hợp bộ định vị là một chương trình khá độc lập với các phần công cụ khác trong hệ thống phần mềm phát triển. Tuy nhiên trong các bộ công cụ phát triển GNU chức năng này được tích hợp luôn trong bộ liên kết. Tuy nhiên không nên nhầm lẫn về chức năng của chúng trong quá trình thực thi biên dịch. Thông thường chương trình chạy trên các máy tính mục đích chung thì hệ điều hành sẽ thực hiện việc chuyển đổi và gán chính xác địa chỉ thực cho các phần mã và dữ liệu trong chương trình ứng dụng, còn với chương trình phát triển chạy trên hệ nhúng thì việc này phải được thực hiện bởi bộ định vị. Đây cũng chính là điểm khác biệt cơ bản khi thực hiện biên dịch một chương trình ứng dụng cho hệ nhúng.

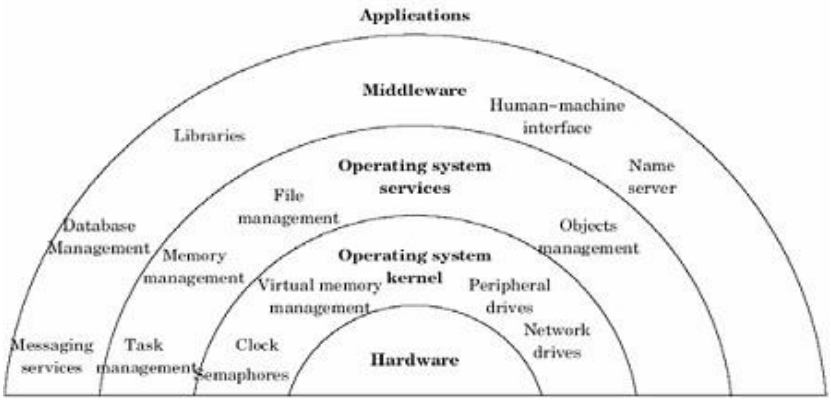
Thông tin về bộ nhớ vật lý của hệ thống phần cứng phát triển mà cần phải cung cấp cho bộ định vị GNU phải được định dạng theo kiểu biểu diễn của bộ liên kết. Thông tin này đôi khi được sử dụng để điều khiển một cách chính xác thứ tự trong các phần mã chương trình và dữ liệu bên trong chương trình có thể định vị lại. Nhưng ở đây chúng ta cần phải thực hiện nhiều hơn thế, tức là phải thiết lập chính xác khu vực của mỗi phần trong bộ nhớ.

Hệ điều hành cho các hệ thống nhúng (HĐH thời gian thực)

Đặc điểm chung của Hệ điều hành

Nguồn gốc ra đời của hệ điều hành là để đảm nhiệm vai trò trung gian để tương tác trực tiếp với phần cứng của máy tính, phục vụ cho nhiều ứng dụng đa dạng. Các hệ điều hành cung cấp một tập các chức năng cần thiết để cho phép các gói phần mềm điều khiển phần cứng máy tính mà không cần phải can thiệp trực tiếp sâu. Hệ điều hành của máy tính có thể thấy nó bao gồm các drivers cho các ngoại vi tích hợp với máy tính như card màn hình, card âm thanh... Các công cụ để quản lý tài nguyên như bộ nhớ và các thiết bị ngoại vi nói chung. Điều này tạo ra một giao diện rất thuận lợi cho các ứng dụng và người sử dụng phát triển phần mềm trên các nền phần cứng đã có. Đồng thời tránh được yêu cầu và hiểu biết sâu sắc về phần cứng và có thể phát triển dựa trên các ngôn ngữ bậc cao.

Hệ thống điều hành bản chất cũng là một loại phần mềm nhưng nó khác với các loại phần mềm thông thường. Sự khác biệt điển hình là hệ thống điều hành được nạp và thực thi đầu tiên khi hệ thống bắt đầu khởi động và được thực hiện trực tiếp bởi bộ xử lý của hệ thống. Hệ thống điều hành được viết để phục vụ điều khiển bộ xử lý cũng như các tài nguyên khác trong hệ thống bởi vì nó sẽ đảm nhiệm chức năng quản lý và lập lịch các quá trình sử dụng CPU và cùng chia sẻ tài nguyên.



Kiến trúc Hệ điều hành

Hệ điều hành cho các hệ thống nhúng - hệ điều hành thời gian thực

Thời gian thực (Real - Time) là gì?

Thời gian thực rất khó định nghĩa. Ý tưởng cơ bản của thời gian thực thể hiện ở chỗ: một hệ thống phải có những phản ứng thích hợp, đúng thời điểm với môi trường của nó. Nhiều người luôn nghĩ rằng, thời gian thực có nghĩa là thực sự nhanh, càng nhanh càng tốt, điều này là sai lầm. Thời gian thực có nghĩa *"đủ nhanh"* (fast enough) trong một ngữ cảnh, một môi trường mà hệ thống đang hoạt động. Khi chúng ta đề cập đến máy tính điều khiển động cơ ô tô, chúng ta cần nó chạy càng nhanh càng tốt.

Một ví dụ khác, khi chúng ta đề cập đến một nhà máy lọc dầu hoá học chẳng hạn, nhà máy được điều khiển bởi một hoặc một hệ thống máy tính. Các máy tính này có trách nhiệm điều khiển quá trình hoá học đồng thời phải phát hiện ra được các sự cố có thể xảy ra. Tuy nhiên, các phản ứng hay các quá trình hoá học thường có hằng số thời gian khá lớn từ hàng giây cho tới hàng phút là ít. Chính vì thế mày chúng ta có thể giả thiết rằng máy tính hoàn toàn có khả năng phản ứng lại các sự cố nghiêm trọng. Tuy nhiên, đặt vấn đề là nếu hệ thống máy tính đó đang trong quá trình in một bản báo cáo dài về các thông số sản lượng của tuần trước thì đột nhiên trục trặc xảy ra. Vậy thì nó mất bao nhiêu thời gian để có thể phản ứng lại các sự cố như thế?

Thực chất của việc tính toán thời gian thực không chỉ ở việc phản ứng đủ nhanh mà còn phải đáng tin cậy và chính xác. Máy tính điều khiển động cơ trong ô tô của bạn phải có thể điều chỉnh luồng nhiên liệu và thời gian đánh lửa một cách hợp lý trong mỗi vòng quay. Nếu không, động cơ sẽ không làm việc theo mong muốn. Máy tính trong nhà máy lọc dầu phải có thể phát hiện và phản ứng lại các điều kiện bất thường trong thời gian cho phép để có thể tránh được các thảm hoạ có thể xảy ra.

Như vậy, nghệ thuật của lập trình thời gian thực chính là việc thiết kế hệ thống sao cho nó có thể tiếp nhận một cách chính xác các ràng buộc về mặt thời gian trong suốt quá trình các sự kiện ngẫu nhiên và không đồng bộ xảy ra.

Các dạng thời gian thực

Về cơ bản, chương trình có tính thời gian thực phải có khả năng phản ứng lại các *sự kiện* trong môi trường mà hệ thống làm việc trong khoảng thời gian nhất định cho trước. Những hệ thống như vậy được gọi là hệ thống *"điều khiển sự kiện"* (hay hệ thống lái sự kiện – *event-driven*) và có thể được mô tả bằng thời gian trễ từ khi mà sự kiện xảy ra cho tới khi hệ thống có hoạt động phản ứng lại với sự kiện đó.

Thời gian thực, mặt khác, đòi hỏi một giới hạn cao hơn về thời gian trễ, được gọi là "*thời hạn lập danh mục*" (*scheduling deadline*). Một hệ thống thời gian thực có thể được chia làm 2 loại: "*thời gian thực cứng*" (*hard real-time*) và "*thờigian thực mềm*" (*soft*

real-time). Trong hệ thống *hard real-time*, hệ thống phải tiếp nhận và nắm bắt được *scheduling deadline* của nó tại mỗi và mọi thời điểm. Sự sai sót trong việc tiếp nhận *deadline* có thể dẫn đến hậu quả nghiêm trọng thậm chí chết người. Lấy ví dụ: máy hỗ trợ nhịp tim cho bệnh nhân khi phẫu thuật. Thuật toán điều khiển phụ thuộc vào thời gian nhịp tim của người bệnh, nếu thời gian này bị trễ, tính mạng của người bệnh sẽ bị ảnh hưởng.

Đối với khái niệm *soft real-time*, *scheduling deadline* có dễ thở hơn chút ít. Chúng ta mong muốn hệ thống phản ứng lại các sự kiện trong thời gian cho phép nhưng không có gì thực sự nghiêm trọng xảy ra nếu hệ thống thỉnh thoảng bị trễ. Lỗi về mặt thời gian có thể chỉ đơn giản là dẫn đến hậu quả giảm độ tin cậy của đối tượng đối với hệ thống mà không có hậu quả thê thảm nào khác xảy ra. Mạng lưới thu ngân tự động của ngân hàng là ví dụ rõ nhất cho *soft real-time*. Mạng rút tiền tự động ATM là hệ thống thời gian thực? Chẳng ai dám đặt cược cả. Khi bạn đưa thẻ ATM vào máy, bạn mong là máy sẽ phản ứng lại trong vòng 1 hay 2 giây. Nhưng nếu nó lâu hơn thế, điều tồi tệ nhất có thể xảy ra là… bạn sốt ruột và thấy khó chịu đối với cái máy đó.

Trên thực tế có rất nhiều hệ thống phối hợp cả 2 loại trên, trong đó, một phần nào đó của hệ thống làm việc dựa trên *hard real-time*, một số phần khác lại dựa trên *soft real-time*.

Hệ điều hành thời gian thực.

Hệ điều hành thời gian thực – RealTime Operating Systems(RTOS), là phần mềm điều khiển chuyên dụng thường được dùng trong những ứng dụng điện toán nhúng có tài nguyên bộ nhớ hạn chế và yêu cầu ngặt nghèo về thời gian đáp ứng tức thời, tính sẵn sàng cao và khả năng tự kiểm soát một cách chính xác.

Có thể tìm thấy RTOS bất kỳ nơi nào. Chúng cũng phổ biến như những hệ điều hành mà bạn đã quen thuộc như Windows, Mac OS và Unix. RTOS âm thầm làm việc bên trong các bộ định tuyến và chuyển mạch trên mạng, động cơ xe, máy nhắn tin, điện thoại di động, thiết bị y tế, thiết bị đo lường và điều khiển công nghiệp và các vô số ứng dụng khác.

Một thuộc tính quan trọng của RTOS là khả năng tách biệt với ứng dụng, vì vậy nếu có một chương trình bị "chết" hay hoạt động không hợp lệ, RTOS có thể nhanh chóng cô lập chương trình này, kích hoạt cơ chế phục hồi và bảo vệ các chương trình khác hay chính bản thân hệ điều hành khỏi các hậu quả của các lệnh sai. Cơ chế bảo vệ tương tự cũng được áp dụng để tránh tình trạng tràn bộ nhớ do bất kỳ chương trình nào gây ra. RTOS xuất hiện ở hai dạng: cứng và mềm. Nếu tính năng xử lý ứng với một sự kiện nào đó không xảy ra hay xảy ra không đủ nhanh, RTOS cứng sẽ chấm dứt hoạt động này và giữ không gây ảnh hưởng đến độ tin cậy và tính sẵn sàng của phần còn lại của hệ thống.

Vì RTOS và máy tính nhúng trở nên phổ biến trong các ứng dụng quan trọng, các nhà phát triển thương mại đang tạo nên những RTOS mới với tính sẵn sàng cao. Những sản phẩm này có một thành phần phần mềm chuyên dụng làm chức năng cảnh báo, chạy các chương trình chẩn đoán hệ thống để giúp xác định chính xác vấn đề trục trặc hay tự động chuyển đổi sang hệ thống dự phòng. Hiện thời RTOS sẵn sàng cao hỗ trợ bus *Compact PCI* của tổ chức *PCI Industrial Computer Manufacturers Group*, bus này dùng cho phần cứng có thể trao đổi nóng.

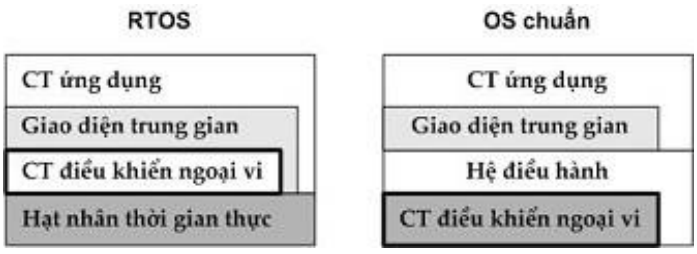
RTOS có rất nhiều dạng. Sản phẩm thương mại như VxWorks và VxWorks AE, đều của Wind River Systems Inc; VxWorks AE được thiết kế với tính sẵn sàng cao, hỗ trợ khả năng gửi thông điệp phân tán và có thể chịu lỗi. RTOS cho phép lập trình viên tách biệt thư viện dùng chung, dữ liệu và phần mềm hệ thống cũng như ứng dụng.

LynxOS là loại RTOS cứng, làm việc với Unix và Java. QNX chạy trên bộ xử lý Intel x86 với nhân chỉ có 10 KB.

RTOS của giới nghiên cứu gồm có Chimera của Đại học Carnegie Mellon. Đây là hệ thống đa nhiệm, đa bộ xử lý thời gian thực, được thiết kế để tạo sự dễ dàng cho các nhà lập trình trong việc tái cấu hình và tái sử dụng mã. Chimera nhắm vào các hệ thống rô bô và tự động. RTOS của Đại học Maryland, có tên là Maruti, hỗ trợ cho cả ứng dụng thời gian thực cứng và mềm.

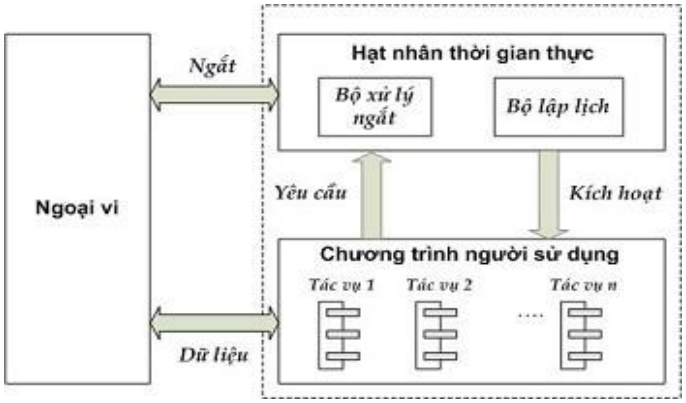
Trong nhiều năm, ứng dụng dựa trên RTOS chủ yếu là trong các hệ thống nhúng và mới gần đây thì chúng đã có mặt khắp nơi, từ thiết bị y tế được điều khiển bằng máy ảnh cho đến máy pha cà phê, những ứng dụng tính toán phân tán đang thúc đẩy các nhà phát triển hệ điều hành thực hiện nghiên cứu và phát triển chuẩn. Chính phủ Mỹ cũng có một số chương trình về lĩnh vực này như công nghệ quản lý tài nguyên thời gian thực, mạng, quản lý dữ liệu và phần mềm điều khiển trung gian. Mục đích của chương trình là làm cho các hệ thống cộng tác, phân tán có thể giao tiếp và chia sẻ tài nguyên với nhau. Một uỷ ban chuyên trách đang đẩy mạnh việc tạo ra khung công nghệ cho tính toán phân tán thời gian thực, áp dụng cho cả ứng dụng quân sự và thương mại. Khung công nghệ này sẽ hỗ trợ các giao tiếp và thành phần liên tác chuẩn.

Cho dù ai là người tạo ra môi trường tính toán phân tán thời gian thực, phổ dụng thì RTOS vẫn sẽ là một trong những công nghệ quan trọng nhất mà người dùng cuối chưa từng nghe đến.



Hệ thống điều hành với phần lõi là hạt nhân phải đảm nhiệm các tác vụ chính như sau:

- Xử lý ngắt
- Lưu trữ ngữ cảnh chương trình tại thời điểm xuất hiện ngắt
- Nhận dạng và lựa chọn đúng bộ xử lý và phục vụ dịch vụ ngắt
- Điều khiển quá trình
- Tạo và kết thúc quá trình/tác vụ
- Lập lịch và điều phối hoạt động hệ thống
- Định thời
- Điều khiển ngoại vi
- Xử lý ngắt
- Khởi tạo giao tiếp vào ra



Cấu trúc Hệ điều hành thời gian thực

Tùy theo cơ chế thực hiện và xây dựng hoạt động của hạt nhân người ta phân loại một số loại hình:

- Hệ thống thời gian thực nhỏ: Với loại này các phần mềm được phát triển mà không cần có hệ điều hành, người lập trình phải tự quản lý và xử lý các vấn đề về điều khiển hệ thống bao gồm:
- Xử lý ngắt
- Điều khiển quá trình/ tác vụ
- Quản lý bộ nhớ
- Công nghệ đa nhiệm
- Mỗi quá trình có một không gian bộ nhớ riêng
- Các quá trình phải được chia nhỏ thành các Thread cùng chia sẻ không gian bộ nhớ.
- Các dịch vụ cung cấp bởi hạt nhân
- Tạo và kết thúc quá trình/ tác vụ
- Truyền thống giữa các quá trình
- Các dịch vụ về định thời gian

- Một số các dịch vụ cung cấp hỗ trợ việc thực thi liên quan đến điều khiển hệ thống

Cơ bản về lập trình nhúng

Biểu diễn số và dữ liệu

- Đơn vị cơ bản nhất trong biểu diễn thông tin của hệ thống số được gọi là *bit,* chính là ký hiệu viết tắt của thuật ngữ *binary digit*.
- Năm 1964, IBM đã thiết kế và chế tạo máy tính số sử dụng một nhóm 8 bit để đánh địa chỉ bộ nhớ và định nghĩa ra thuật ngữ *8 bit = 1 byte*.
- Ngày nay sử dụng rộng rãi thuật ngữ *word* là một từ dữ liệu dùng để biểu diễn kích thước dữ liệu mà được xử lý một cách hiệu quả nhất đối với mỗi loại kiến trúc xử lý số cụ thể. Chính vì vậy một từ có thể là 16 bits, 32 bits, hoặc 64 bits…
- Mỗi một byte có thể được chia ra thành hai nửa 4 bit và được gọi là các *nibble*. *Nibble* chứa các bit trọng số lớn được gọi là *nibble* bậc cao, và *nibble* chứa các bit trọng số nhỏ được gọi là *nibble* bậc thấp.

Các hệ thống cơ số

Trong các hệ thống biểu diễn số hiện nay đều được biểu diễn ở dạng tổng quát là tổng luỹ thừa theo cơ số, và được phân loại theo giá trị cơ số. Một cách tổng quát một hệ biểu diễn số cơ số b và a được biểu diễn như sau:

$A=a_{\mathbf{n}}b^{\mathbf{n}}+ a_{\mathbf{n-1}}b^{\mathbf{n-1}}+\ldots+ a_0 = \sum_{i=0}^{n} ai*bi$

Ví dụ: như cơ số *binary* (nhị phân), cơ số *decimal* (thập phân, cơ số 8 *Octal* (bát phân).

Ví dụ về biểu diễn các giá trị trong các hệ cơ số khác nhau:

- $243.51_{\mathbf{10}} = 2 \times 10^{\mathbf{2}} + 4 \times 10^{\mathbf{1}} + 3 \times 10^{\mathbf{0}} + 5 \times 10^{\mathbf{-1}} + 1 \times 10^{\mathbf{-2}}$
- $212_{\mathbf{3}} = 2 \times 3^{\mathbf{2}} + 1 \times 3^{\mathbf{1}} + 2 \times 3^{\mathbf{0}}$
- $10110_{\mathbf{2}} = 1 \times 2^{\mathbf{4}} + 0 \times 2^{\mathbf{3}} + 1 \times 2^{\mathbf{2}} + 1 \times 2^{\mathbf{1}} + 0 \times 2^{\mathbf{0}} = 22_{\mathbf{10}}$

Hai loại cơ số biểu diễn thông dụng nhất hiện nay cho các hệ thống xử lý số là *cơ số nhị phân* và *cơ số mười sáu*.

Số nguyên

Trong biểu diễn số có dấu để phân biệt số dương và số âm người ta sử dụng bit trọng số lớn nhất qui ước làm bit dấu và các bit còn lại được sử dụng để biểu diễn giá trị độ lớn của số. Ví dụ một từ 8 bit được sử dụng để biểu diễn giá trị -1 sẽ có dạng nhị phân là 10000001, và giá trị +1 sẽ có dạng 00000001. Như vậy với một từ 8 bit có thể biểu diễn

được các số trong phạm vi từ -127 đến +127. Một cách tổng quát một từ *N* bit sẽ biểu diễn được *-2(N-1)-1* đến *+2(N-1)-1*.

Chú ý khi thực hiện cộng hai số có dấu:

- Nếu hai số cùng dấu thì thực hiện phép cộng phần biểu diễn giá trị và sử dụng bit dấu cùng dấu với hai số đó.
- Nếu hai số khác dấu thì kết quả sẽ nhận dấu của toán tử lớn hơn, và thực hiện phép trừ giữa toán tử có giá trị lớn hơn với toán tử bé hơn.

Ví dụ 1: Cộng hai số có dấu 010011111**2** và 001000111**2**.

1 1 1 1 ⇐carries (Số nhớ)

0 1 0 0 1 1 1 1 (79)

0 + <u>0 1 0 0 0 1 1</u> + (35)

0 1 1 1 0 0 1 0 (114)

Ví dụ 2: Trừ hai số có dấu 010011111**2** và 011000111**2**.

0 1 1 2 ⇐borrows (số vay)

0 1 1 0 0 0 1 1 (99)

0 - <u>1 0 0 1 1 1 1</u> - (79)

0 0 0 1 0 1 0 0 (20)

- Thuật toán thực hiện phép tính có dấu:
- (1) Khai báo và xóa các biến lưu giá trị và dấu để chuẩn bị thực hiện phép tính.
- (2) Kiểm tra dấu của toán tử thứ nhất để xem có phải số âm không. Nếu là số âm thì thực hiện bù dấu và bù toán tử. Nếu không thì chuyển qua thực hiện bước 3.
- (3) Kiểm tra dấu của toán tử thứ hai để xem có phải số âm không. Nếu là số âm thì thực hiện bù dấu và bù toán tử. Nếu không thì chuyển sang thực hiện bước 4.
- (4) Thực hiện phép nhân hoặc chia với các toán tử vừa xử lý.
- (5) Kiểm tra dấu. Nếu zero thì coi như đã kết thúc. Nếu bằng -1 (0ffh) thì thực hiện phép tính bù hai với kết quả thu được và kết thúc.

Hiện nay người ta sử dụng hai qui ước biểu diễn số nguyên phân biệt theo thứ tự của byte trọng số trong một từ được biểu diễn:

- *Litte edian*: byte trọng số nhỏ nhất đứng trước thuận lợi cho phép cộng hoặc trừ.
- *Big endian*: byte trọng số lớn nhất đứng trước thuận lợi cho phép nhân hoặc chia.

Ví dụ: xét một số nhị phân 4*byte*

Byte 3	Byte 2	Byte 1	Byte 0

Theo qui ước biểu diễn *litte edian* thì thứ tự địa chỉ lưu trong bộ nhớ sẽ là:

Địa chỉ cở sở + 0 = Byte 0

Địa chỉ cơ sở + 1 = Byte 1

Địa chỉ cơ sở + 2 = Byte 2

Địa chỉ cơ sở + 3 = Byte 3

Và theo qui ước biểu diễn số *big edian* sẽ là:

Địa chỉ cở sở + 0 = Byte 3

Địa chỉ cơ sở + 1 = Byte 2

Địa chỉ cơ sở + 2 = Byte 1

Địa chỉ cơ sở + 3 = Byte 0

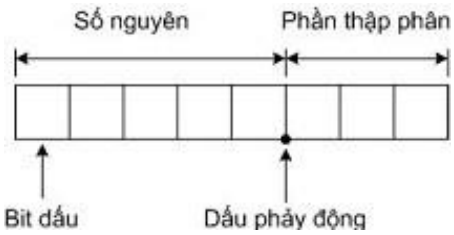
Số dấu phẩy tĩnh

Chúng ta có thể sử dụng một ký hiệu dấu chấm ảo để biểu diễn một số thực. Dấu chấm ảo được sử dụng trong từ dữ liệu dùng để phân biệt và ngăn cách giữa phần biểu diễn giá trị nguyên của dữ liệu và một phần lẻ thập phân. Ví dụ về một từ 8bit biểu diễn số dấu phảy động được chỉ ra như trong Hình 6.1. Với cách biểu diễn này, giá trị thực của số được tính như sau:

$N = a_{\mathbf{4}}\, 2_{\mathbf{4}} + a_{\mathbf{3}}\, 2_{\mathbf{3}} + a_{\mathbf{2}}\, 2_{\mathbf{2}} + a_{\mathbf{1}}\, 2_{\mathbf{1}} + a_{\mathbf{0}}\, 2_{\mathbf{0}} + a_{\mathbf{-1}}\, 2_{\mathbf{-1}} + a_{\mathbf{-2}}\, 2_{\mathbf{-2}} + a_{\mathbf{-3}}\, 2_{\mathbf{-3}}$

$= 0.2_{\mathbf{4}} + 1.2_{\mathbf{3}} + 0.2_{\mathbf{2}} + 1.2_{\mathbf{1}} + 0.2_{\mathbf{0}} + 1.2_{\mathbf{-1}} + 0.2_{\mathbf{-2}} + 1.2_{\mathbf{-3}}$

= 8+2+1+1/2+1/8 = 11.625



Định dạng biểu diễn số dấu phảy tĩnh 8 bit

Nhược điểm của phương pháp biểu diễn số dấu phảy tĩnh là vùng biểu diễn số nguyên bị hạn chế bởi dấu phảy tĩnh được gán cố định. Điều này dễ xảy ra hiện tượng tràn số khi thực hiện các phép nhân hai số lớn.

Số dấu phẩy động

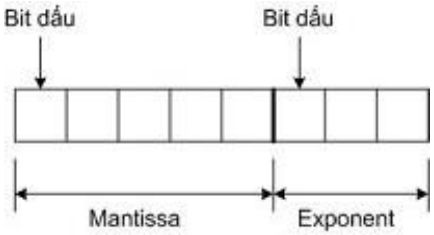
Phương pháp biểu diễn số chính xác và linh hoạt được sử dụng rộng rãi hiện nay là hệ thống biểu diễn số dấu phảy động. Đây cũng là một phương pháp biểu diễn số khoa học bao gồm 2 phần: phần biểu diễn lưu trữ số *mantissa* và một phần lưu trữ biểu diễn số *exponent*. Ví dụ trong hệ cơ số thập phân, một số nguyên bằng 5 có thể được biểu diễn hoặc là 0.5*10**1** , 50 * 10−**1** , hoặc 0.05*10**2** …Trong máy tính số hoặc hệ thống số nói chung, các số dấu phảy động nhị phân thường được biểu diễn dạng:

$N = M * 2^{\mathbf{E}}$

Trong đó:

- M là phần giá trị số *mantissa*,
- E là phần lũy thừa của số N. M thường là các giá trị lẻ mà phần thập phân của nó thường nằm trong khoảng $0.5 \leq M \leq 1$.

Hình 6.2 mô tả biểu diễn một số dấu phảy động của từ 8 bit gồm 5 bit biểu diễn phần số có nghĩa *mantissa*, và 3 bit biểu diễn phần lũy thừa. Vì các phần *mantissa* và lũy thừa đều có thể nhận các giá trị âm vì vậy các bit đầu tiên của các phần giá trị đó đều có thể được sử dụng để biểu diễn dấu khi cần thiết.



Biểu diễn dấu phảy động 8bit

Trong một số VXL, VĐK do độ rộng từ nhị phân nhỏ nên có thể sử dụng 2 từ để biểu diễn một số dấu phảy động. Một từ sẽ dùng để biểu diễn giá trị *mantissa*, và một phần biểu diễn giá trị *exponent*.

Nếu phần *mantissa* được chuẩn hóa thành một số lẻ có giá trị trong khoảng $0.5 \leq M \leq 1$ thì bit đầu tiên sau bit dấu thường là một và sẽ có một dấu phảy nhị phân ẩn ngay sau bit dấu.

Phần biểu diễn *exponent* E sẽ quyết định vị trí của dấu phảy động sẽ dịch sang trái (E>0) hay sang phải (E<0) bao nhiêu vị trí. Ví dụ biểu diễn một số thập phân 6.5 bằng một từ 8 bit dấu phảy động như sau:

*N = .1101 * 2* **11(2)** *= (1/2+1/4+1/16)*2 3 = 6.5*

Trong trường hợp này phần *mantissa* gồm 4 bit và phần *exponent* gồm 3 bit. Nếu ta dịch dấu phảy sang phải 3 vị trí bit thì chúng ta sẽ có một số nhị phân dấu phảy động biểu diễn được sẽ là *110.1*.

Tổng quát hóa trong trường hợp một số nhị phân dấu phảy động n bit gồm m bit biểu diễn phần *mantissa* và *e* bit biểu diễn phần *exponent* thì giá trị của số lớn nhất có thể biểu diễn được sẽ là:

$$N_{\max} = (1 - 2^{-m+1})2^{(2^{e-1}-1)}$$

Và số dương nhỏ nhất có thể biểu diễn là:

$$N_{\min} = 0.5 * 2^{-(2^{e-1}-1)}$$

Theo tiêu chuẩn IEEE 754 và 854 có 2 định dạng chính cho số dấu phảy động là số thực dài (*long*) và số thực ngắn (*short*) chúng khác nhau về dải biểu diễn và độ lớn lưu trữ yêu cầu. Theo chuẩn này, số thực dài được định dạng 8 byte bao gồm 1 bit dấu, 11 bit *exponent* và 53 bit lưu giá trị số có nghĩa. Một số thực ngắn được định dạng 4 byte bao gồm 1 bit dấu, 8 bit lũy thừa và 24 bit lưu giá trị số có nghĩa. Một số thực ngắn có thể biểu diễn và xử lý được số có giá trị nằm trong dải 10**38** to 10-**38** và số thực dài có thể biểu diễn và xử lý được số có giá trị thuộc dải 10**308** to 10-**308** . Để biểu diễn một giá trị tương đương như vậy bằng số dấu phảy tĩnh thì cần tới 256 bit hay 32 byte dữ liệu.

Ngôn ngữ lập trình

Một trong những ngôn ngữ lập trình có lẽ phổ cập rộng rãi nhất hiện nay là ngôn ngữ *C*. So với bất kỳ ngôn ngữ lập trình nào khác đang tồn tại C thực sự phù hợp và trở thành một ngôn ngữ phát triển của hệ nhúng. Điều này không phải là cố hữu và sẽ tồn tại mãi, nhưng tại thời điểm này thì C có lẽ là một ngôn ngữ gần gũi nhất để trở thành một chuẩn

ngôn ngữ trong thế giới hệ nhúng. Trong phần này chúng ta sẽ cùng tìm hiểu tại sao C lại trở thành một ngôn ngữ phổ biến đến vậy và tại sao chúng ta lựa chọn nó như một ngôn ngữ minh họa cho việc lập trình hệ nhúng.

Sự thành công về phát triển phần mềm thường là nhờ vào sự lựa chọn ngôn ngữ phù hợp nhất cho một dự án đặt ra. Cần phải tìm một ngôn ngữ để có thể *đáp ứng được yêu cầu lập trình cho các bộ xử lý từ 8bit đến 64bit*, trong các hệ thống chỉ có hữu hạn về bộ nhớ vài Kbyte hoặc Mbyte. Cho tới nay, điều này chỉ có C là thực sự có thể thỏa mãn và phù hợp nhất.

Rõ ràng C có một số ưu điểm nổi bật tiêu biểu như khá nhỏ và *dễ dàng cho việc học, các chương trình biên dịch thường khá sẵn cho hầu hết các bộ xử lý đang sử dụng hiện nay*, và có *rất nhiều người đã biết và làm chủ được ngôn ngữ này rồi*, hay nói cách khác cũng *đã được phổ cập từ lâu*. Hơn nữa C có lợi thế là *không phụ thuộc vào bộ xử lý thực thi mã nguồn*. Người lập trình chỉ phải tập trung chủ yếu vào việc xây dựng thuật toán, ứng dụng và thể hiện bằng ngôn ngữ thân thiện thay vì phải tìm hiểu sâu về kiến thức phần cứng, cũng như rất nhiều các ưu điểm nổi bật khác của ngôn ngữ bậc cao nói chung.

Có lẽ một thế mạnh lớn nhất của C là một ngôn ngữ bậc cao mức thấp nhất. Tức là với ngôn ngữ C chúng ta vẫn có thể điều khiển và truy nhập trực tiếp phần cứng khá thuận tiện mà không hề phải hy sinh hay đánh đổi bất kỳ một thế mạnh nào của ngôn ngữ bậc cao. Thực chất đây cũng là một trong những tiêu chí xây dựng của những người sáng lập ra ngôn ngữ C muốn hướng tới. Thực tế điều này đã được đề cập đến khi hai nhà sáng lập ra ngôn ngữ C, *Kernighan*và*Ritchie* đã đưa vào trong phần giới thiệu của cuốn sách của họ "*The C Programming Language*" như sau: "*C is a relatively "low level" language. This characterization is not pejorative; it simply means that C deals with the same sort of objects that most computers do. These may be combined and moved about with the arithmetic and logical operators implemented by real machines…"*

Tất nhiên là C không phải là ngôn ngữ duy nhất cho các nhà lập trình nhúng. Ít nhất hiện nay người ta cũng có thể biết tới ngoài ngôn ngữ C là Assembly, C++, và Ada.

Trong những buổi đầu phát triển hệ nhúng thì ngôn ngữ Assembly chủ yếu được sử dụng cho các vi xử lý đích. Với ngôn ngữ này cho phép người lập trình điều khiển và kiểm soát hoàn toàn vi xử lý cũng như phần cứng hệ thống trong việc thực thi chương trình. Tuy nhiên ngôn ngữ Assembly có nhiều nhược điểm mà cũng chính là lý do tại sao hiện nay nó ít được phổ cập và sử dụng. Đó là, việc học và sử dụng ngôn ngữ Assembly rất khó khăn và đặc biệt khó khăn trong việc phát triển các chương trình ứng dụng lớn phức tạp. Hiện nay nó chỉ được sử dụng chủ yếu như điểm nối giữa ngôn ngữ bậc cao và bậc thấp và được sử dụng khi có yêu cầu đặc biệt về hiệu suất thực hiện và tối ưu về tốc độ mà không thể đạt được bằng ngôn ngữ khác. Ngôn ngữ Assembly chỉ thực sự phù hợp cho những người có kinh nghiệm và hiểu biết tốt về cấu trúc phần cứng đích cũng như nguyên lý thực hiện của bộ lệnh và chíp xử lý.

C++ là một ngôn ngữ kế thừa từ C để nhằm vào các lớp ứng dụng và tư duy lập trình hướng đối tượng và cũng bắt đầu chiếm được số lượng lớn quan tâm trong việc ứng dụng cho phát triển hệ nhúng. Tất cả các đặc điểm cốt lõi của C vẫn được kế thừa hoàn toàn trong ngôn ngữ C++ và ngoài ra còn hỗ trợ khả năng mới về trừu tượng hóa dữ liệu và phù hợp với tư duy lập trình hiện đại; hướng đối tượng. Tuy nhiên điều này bị đánh đổi bởi hiệu suất và thời gian thực thi do đó chỉ phù hợp với các dự án phát triển chương trình lớn và không chịu sức ép lớn về thời gian thực thi.

Ada cũng là một ngôn ngữ hướng đối tượng mặc dù nó không được phổ cập rộng rãi như C++. Ada được xây dựng bởi cơ quan quốc phòng Mỹ để phục vụ phát triển các phần mềm quân sự chuyên dụng đặc biệt. Mặc dù cũng đã được chuẩn hóa quốc tế (Ada83 và Ada95) nhưng nó vẫn không được phổ cập rộng rãi ngoài việc ứng dụng chủ yếu trong các lĩnh vực quân sự và hàng không vũ trụ. Và nó cũng dần dần bị mất ưu thế và sự phổ cập trong thời gian gần đây, đây cũng là một điều đáng tiếc vì bản thân Ada cũng là một ngôn ngữ có nhiều đặc điểm phù hợp cho việc phát triển phần mềm hệ nhúng thay vì phải sử dụng C++.

Các kiến trúc phần mềm nhúng thông dụng

Hiện nay tồn tại một số loại kiến trúc phần mềm thông dụng trong các hệ thống nhúng như sau:

Vòng lặp kiểm soát đơn giản

Theo thiết kế này, phần mềm được tổ chức thành một vòng lặp đơn giản. Vòng lặp gọi đến các chương trình con, mỗi chương trình con quản lý một phần của hệ thống phần cứng hoặc phần mềm.

Hệ thống ngắt điều khiển

Các hệ thống nhúng thường được điều khiển bằng các ngắt. Có nghĩa là các tác vụ của hệ thống nhúng được kích hoạt bởi các loại sự kiện khác nhau. Ví dụ, một ngắt có thể được sinh ra bởi một bộ định thời sau một chu kỳ được định nghĩa trước, hoặc bởi sự kiện khi cổng nối tiếp nhận được một byte nào đó.

Loại kiến trúc này thường được sử dụng trong các hệ thống có bộ quản lý sự kiện đơn giản, ngắn gọn và cần độ trễ thấp. Hệ thống này thường thực hiện một tác vụ đơn giản trong một vòng lặp chính. Đôi khi, các tác vụ phức tạp hơn sẽ được thêm vào một cấu trúc hàng đợi trong bộ quản lý ngắt để được vòng lặp xử lý sau đó. Lúc này, hệ thống gần giống với kiểu nhân đa nhiệm với các tiến trình rời rạc.

Đa nhiệm tương tác

Một hệ thống đa nhiệm không ưu tiên cũng gần giống với kỹ thuật vòng lặp kiểm soát đơn giản ngoại trừ việc vòng lặp này được ẩn giấu thông qua một giao diện lập trình API. Các nhà lập trình định nghĩa một loạt các nhiệm vụ, mỗi nhiệm vụ chạy trong một môi trường riêng của nó. Khi không cần thực hiện nhiệm vụ đó thì nó gọi đến các tiến trình con tạm nghỉ (bằng cách gọi "pause", "wait", "yeild" …).

Ưu điểm và nhược điểm của loại kiến trúc này cũng giống với kiểm vòng lặp kiểm soát đơn giản. Tuy nhiên, việc thêm một phần mềm mới được thực hiện dễ dàng hơn bằng cách lập trình một tác vụ mới hoặc thêm vào hàng đợi thông dịch (queue-interpreter).

Đa nhiệm ưu tiên

Ở loại kiến trúc này, hệ thống thường có một đoạn mã ở mức thấp thực hiện việc chuyển đổi giữa các tác vụ khác nhau thông qua một bộ định thời. Đoạn mã này thường nằm ở mức mà hệ thống được coi là có một hệ điều hành và vì thế cũng gặp phải tất cả những phức tạp trong việc quản lý đa nhiệm.

Bất kỳ tác vụ nào có thể phá hủy dữ liệu của một tác vụ khác đều cần phải được tách biệt một cách chính xác. Việc truy cập tới các dữ liệu chia sẻ có thể được quản lý bằng một số kỹ thuật đồng bộ hóa như hàng đợi thông điệp (message queues), semaphores … Vì những phức tạp nói trên nên một giải pháp thường được đưa ra đó là sử dụng một hệ điều hành thời gian thực. Lúc đó, các nhà lập trình có thể tập trung vào việc phát triển các chức năng của thiết bị chứ không cần quan tâm đến các dịch vụ của hệ điều hành nữa.

Vi nhân (Microkernel) và nhân ngoại (Exokernel)

Khái niệm vi nhân (microkernel) là một bước tiếp cận gần hơn tới khái niệm hệ điều hành thời gian thực. Lúc này, nhân hệ điều hành thực hiện việc cấp phát bộ nhớ và chuyển CPU cho các luồng thực thi. Còn các tiến trình người dùng sử dụng các chức năng chính như hệ thống file, giao diện mạng lưới,… Nói chung, kiến trúc này thường được áp dụng trong các hệ thống mà việc chuyển đổi và giao tiếp giữa các tác vụ là nhanh.

Còn nhân ngoại (exokernel) tiến hành giao tiếp hiệu quả bằng cách sử dụng các lời gọi chương trình con thông thường. Phần cứng và toàn bộ phần mềm trong hệ thống luôn đáp ứng và có thể được mở rộng bởi các ứng dụng.

Nhân khối (monolithic kernels)

Trong kiến trúc này, một nhân đầy đủ với các khả năng phức tạp được chuyển đổi để phù hợp với môi trường nhúng. Điều này giúp các nhà lập trình có được một môi trường giống với hệ điều hành trong các máy để bàn như Linux hay Microsoft Windows và vì thế rất thuận lợi cho việc phát triển. Tuy nhiên, nó lại đòi hỏi đáng kể các tài nguyên phần cứng làm tăng chi phí của hệ thống. Một số loại nhân khối thông dụng là Embedded Linux và Windows CE. Mặc dù chi phí phần cứng tăng lên nhưng loại hệ thống nhúng này đang tăng trưởng rất mạnh, đặc biệt là trong các thiết bị nhúng mạnh như Wireless router hoặc hệ thống định vị GPS. Lý do của điều này là:

- Hệ thống này có cổng để kết nối đến các chip nhúng thông dụng.
- Hệ thống cho phép sử dụng lại các đoạn mã sẵn có phổ biến như các trình điều khiển thiết bị, Web Servers, Firewalls, …
- Việc phát triển hệ thống có thể được tiến hành với một tập nhiều loại đặc tính, chức năng còn sau đó lúc phân phối sản phẩm, hệ thống có thể được cấu hình để loại bỏ một số chức năng không cần thiết. Điều này giúp tiết kiệm được những vùng nhớ mà các chức năng đó chiếm giữ.
- Hệ thống có chế độ người dùng để dễ dàng chạy các ứng dụng và gỡ rối. Nhờ đó, qui trình phát triển được thực hiện dễ dàng hơn và việc lập trình có tính linh động hơn.
- Có nhiều hệ thống nhúng thiếu các yêu cầu chặt chẽ về tính thời gian thực của hệ thống quản lý. Còn một hệ thống như Embedded Linux có tốc độ đủ nhanh để trả lời cho nhiều ứng dụng. Các chức năng cần đến sự phản ứng nhanh cũng có thể được đặt vào phần cứng.

Tập lệnh

Cấu trúc tập lệnh CISC và RISC

Hầu hết các vi điều khiển và VXL nhúng có cấu trúc được phát triển dựa theo kiến trúc máy tính tập lệnh phức hợp *CISC (Complex Instruction Set Computer).* CISC là một cấu trúc xử lý các lệnh phức hợp, tức là một lệnh phức hợp sẽ bao gồm một vài lệnh đơn. Theo nguyên lý này có thể giảm bớt được thời gian dùng để truy nhập và đọc mã chương trình từ bộ nhớ. Điều này rất có ý nghĩa với các kiến trúc thiết kế xử lý tính toán theo kiểu tuần tự. Lý do cho sự ra đời của tập lệnh phức hợp nhằm giảm thiểu dung lượng bộ nhớ cần thiết để lưu giữ chương trình thực hiện, và sẽ giảm được giá thành về bộ nhớ cần cung cấp cho CPU. Các lệnh càng gọn và phức hợp thì sẽ cần càng ít không gian bộ nhớ chương trình. Kiến trúc tập lệnh phức hợp sử dụng các lệnh với độ dài biến đổi tuỳ thuộc vào độ phức hợp của các lệnh từ đơn giản đến phức tạp. Trong đó sẽ có một số lượng lớn các lệnh có thể truy nhập trực tiếp bộ nhớ. Vì vậy với kiến trúc tập lệnh phức hợp chúng ta sẽ có được một tập lệnh đa dạng phức hợp, gọn, với độ dài lệnh thay đổi và dẫn đến chu kỳ thực hiện lệnh cũng thay đổi tuỳ theo độ phức hợp trong từng lệnh.

Một vài lệnh phức hợp, đặc biệt là các lệnh truy nhập bộ nhớ cần tới vài chục chu kỳ để thực hiện. Trong một số trường hợp các nhà thiết kế VXL thấy rằng cần phải giảm chu kỳ nhịp lệnh để có đủ thời gian cho các lệnh hoàn thành điều này cũng dẫn đến thời gian thực hiện bị kéo dài hơn.

Một số VĐK được phát triển theo kiến trúc máy tính tập lệnh rút gọn *RISC (Reduced Instruction Set Computer)*. RISC phù hợp với các thiết kế kiến trúc xử lý các lệnh đơn. Thuật ngữ "rút gọn" (reduced) đôi khi bị hiểu không thật chính xác theo nghĩa đen của nó thực chất ý tưởng gốc xuất phát từ khả năng cung cấp một tập lệnh tối thiểu để thực hiện tất cả các hoạt động chính như: chuyển dữ liệu, các hoạt động *ALU* và rẽ nhánh điều khiển chương trình. Chỉ có các lệnh nạp (*load*), lữu trữ (*store*) là được phép truy nhập trực tiếp bộ nhớ.

CISC	RISC
Bất kỳ lệnh nào cũng có thể tham chiếu tới bộ nhớ	Chỉ có các lệnh Nạp (Load) hoặc Lưu trữ (Store) là có thể tham chiếu tới bộ nhớ
Tồn tại nhiều lệnh và kiểu địa chỉ	Tồn tại ít lệnh và kiểu địa chỉ
Khuôn dạng lệnh đa dạng	Khuôn dạng lệnh cố định
Chỉ có một tập thanh ghi	Có nhiều tập thanh ghi
Các lệnh thực hiện trong nhiều nhịp chu kỳ	Các lệnh thực hiện trong một nhịp chu kỳ
Có một chương trình nhỏ để thông dịch lệnh	Lệnh được thực hiện trực tiếp ngay bởi phần cứng
Chương trình thông dịch lệnh phức tạp	Chương trình biên dịch mã nguồn phức tạp
Không hỗ trợ cơ chế pipeline	Hỗ trợ cơ chế pipeline
Kích thước mã chương trình nhỏ gọn	Kích thước mã chương trình lớn

So sánh đặc điểm của CISC và RISC

Các kiểu truyền địa chỉ toán tử lệnh

Các kiểu đánh/truyền địa chỉ cho phép chúng ta chỉ ra/truyền toán tử tham gia trong các lệnh thực thi. Kiểu địa chỉ có thể chỉ ra là một hằng số, một thanh ghi hoặc một khu vực cụ thể trong bộ nhớ. Một số kiểu đánh địa chỉ cho phép sử dụng địa chỉ ngắn và một số loại thì cho phép chúng ta xác định khu vực chứa toán tử lệnh, và thường được gọi là địa chỉ hiệu dụng của toán tử và thường là động. Chúng ta sẽ xét một số loại hình đánh địa chỉ cơ bản hiện đang được sử dụng rộng rãi trong cơ chế thực hiện lệnh.

- *Đánh địa chỉ tức thì (Immediate Addressing)*: Phương pháp này cho phép truyền giá trị toán tử lệnh một cách tức thì như một phần của câu lệnh được thực thi. Ví dụ nếu sử dụng kiểu đánh địa chỉ tức thời cho câu lệnh Load 0x0008 thì giá trị 0x0008 sẽ được nạp ngay vào AC. Trường bit thường dùng để chứa toán tử lệnh sẽ chứa giá trị thực của toán tử chứ không phải địa chỉ của toán tử cần truyền cho lệnh thực thi. Kiểu địa chỉ tức thời cho phép thực thi

lệnh rất nhanh vì không phải thực hiện truy xuất bộ nhớ để nạp giá trị toán tử mà giá trị toán tử đã được gộp như một phần trong câu lệnh và có thể thực thi ngay. Vì toán tử tham gia như một phần cố định của chương trình vì vậy kiểu đánh địa chỉ này chỉ phù hợp với các toán tử hằng và biết trước tại thời điểm thực hiện chương trình, hay đã xác định tại thời điểm biên dịch chương trình.

- *Đánh địa chỉ trực tiếp (Direct Addressing):*Phương pháp này cho phép truyền toán tử lệnh thông qua địa chỉ trực tiếp chứa toán tử đó trong bộ nhớ. Ví dụ nếu sử dụng cơ chế đánh địa chỉ toán tử trực tiếp thì trong câu lệnh Load 0x0008 sẽ được hiểu là dữ liệu hay toán tử được nạp trong câu lệnh này nằm trong bộ nhớ tại địa chỉ 0x0008. Cơ chế đánh địa chỉ trực tiếp cũng thuộc loại hình khá nhanh mặc dù không nhanh được như cơ chế truyền địa chỉ tức thời nhưng độ mềm dẻo cao hơn vì địa chỉ của toán tử không nằm trong phần mã lệnh và giá trị có thể thay đổi trong quá trình thực thi chương trình.
- *Đánh địa chỉ thanh ghi (Register Addressing)*: Trong cách đánh địa chỉ và truyền toán tử này thì toán tử không nằm trong bộ nhớ như trường hợp đánh địa chỉ trực tiếp mà nằm tại chính trong thanh ghi. Khi toán tử đã được nạp vào thanh ghi thì việc thực hiện có thể rất nhanh vì tốc độ truy xuất thanh ghi nhanh hơn so với bộ nhớ. Nhưng số lượng thanh ghi chỉ có hạn và phải được chia sẻ trong quá trình thực hiện chương trình vì vậy các toán tử phải được nạp vào thanh ghi trước khi nó được thực thi.
- *Đánh địa chỉ gián tiếp (Indirect Addressing)*: Trong phương pháp truyền toán tử này, trường toán tử trong câu lệnh được sử dụng để tham chiếu tới một con trỏ nằm trong bộ nhớ để trỏ tới địa chỉ hiệu dụng của toán tử. Cơ chế truyền này có thể nói là mềm dẻo nhất so với các cơ chế truyền địa chỉ khác trong quá trình thực thi chương trình. Ví dụ nếu áp dụng cơ chế truyền địa chỉ gián tiếp trong câu lệnh Load 0x0008 thì sẽ được hiểu là giá trị dữ liệu có địa chỉ tại 0x0008 thực chất là chứa địa chỉ hiệu dụng của toán tử cần truyền cho câu lệnh. Giả thiết tại vị trí ô nhớ 0x0008 đang chứa giá trị 0x02A0 thì 0x02A0 chính là giá trị thực của toán tử sẽ được nạp vào AC. Một biến thể khác cũng có thể thực hiện theo cơ chế này là truyền tham chiếu tới con trỏ nằm trong khu vực thanh ghi. Cơ chế này còn được biết tới với tên gọi là đánh địa chỉ gián tiếp thanh ghi. Ví dụ một câu lệnh Load R1 sử dụng cơ chế truyền địa chỉ gián tiếp thanh ghi thì chúng ta có thể dễ dàng thông dịch được toán tử truyền trong câu lệnh này có địa chỉ hiệu dụng nằm trong thanh ghi R1.
- *Cách đánh địa chỉ cơ sở và chỉ số (Indexed and Based Addressing):* Trong cơ chế này người ta sử dụng một thanh ghi để chứa offset (độ chênh lệch tương đối) mà sẽ được cộng với toán tử để tạo ra một địa chỉ hiệu dụng. Ví dụ, nếu toán tử X của lệnh Load X được đánh địa chỉ theo cơ chế địa chỉ chỉ số và thanh ghi R1 là thanh ghi chứa chỉ số và có giá trị là 1 thì địa chỉ hiệu dụng của toán tử thực chất sẽ là X+1. Cơ chế đánh địa chỉ cơ sở cũng giống như vậy ngoại trừ một điều là thay vì sử dụng thanh ghi địa chỉ offset thì ở đây sử dụng thanh ghi địa chỉ cơ sở. Về mặt lý thuyết sự khác nhau giữa hai cơ chế tham chiếu địa chỉ này là chúng được sử dụng thế nào chứ không phải các toán tử

được tính toán thể nào. Một thanh ghi chỉ số sẽ lưu chỉ số mà sẽ được sử dụng như một offset so với địa chỉ được đưa ra trong trường địa chỉ của lệnh thực thi. Thanh ghi cơ sở lưu một địa chỉ cơ sở và trường địa chỉ trong câu lệnh thực thi sẽ lưu giá trị dịch chuyển từ địa chỉ này. Hai cơ chế tham chiếu địa chỉ này rất hữu ích trong việc truy xuất với các phần tử kiểu mảng. Tuỳ thuộc vào thiết kế tập lệnh các thanh ghi mục đích chung thường hay được sử dụng trong cơ chế đánh địa chỉ này.

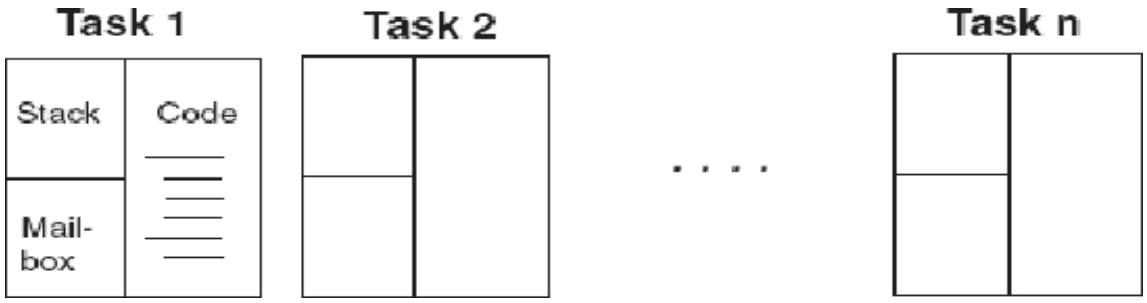
- *Đánh địa chỉ ngăn xếp (Stack Addressing)*: Trong cơ chế truyền địa chỉ này thì toán tử nhận được từ đỉnh ngăn xếp. Thay vì sử dụng thanh ghi mục đích chung hay ô nhớ kiến trúc dựa trên ngăn xếp lưu các toán tử trên đỉnh của ngăn xếp, và có thể truy xuất với CPU. Kiến trúc này không chỉ hiệu quả trong việc lưu giữ các giá trị trung gian trong các phép tính phức tạp mà còn cung cấp một phương pháp hiệu quả trong việc truyền các tham số trong các lời gọi hàm cũng như để lưu cất các cấu trúc dữ liệu cục bộ và định nghĩa ra phạm vi tồn tại của các biến và các hàm con. Trong các cấu trúc lệnh truyền toán tử dựa trên ngăn xếp, hầu hết các lệnh chỉ bao gồm phần mã, tuy nhiên cũng có một số lệnh đặc biệt chỉ có một toán tử ví dụ như lệnh cất vào (push) hoặc lấy ra (pop) từ ngăn xếp. Chỉ có một số lệnh yêu cầu hai toán tử thì hai giá trị chứa trong 2 ô nhớ trên đỉnh ngăn xếp sẽ được sử dụng. Ví dụ như lệnh Add, CPU lấy ra khỏi ngăn xếp hai phần tử nằm trên đỉnh rồi thực hiện phép cộng và sau đó lưu kết quả trở lại đỉnh ngăn xếp.
- *Các cách đánh địa chỉ khác:* Có rất nhiều biến có thể tạo bởi các cơ chế đánh địa chỉ giới thiệu ở trên. Đó là sự tổ hợp trong việc tạo ra hoặc xác định địa chỉ hiệu dụng của toán tử truyền cho lệnh thực thi. Ví dụ như cơ chế đánh địa chỉ chỉ số gián tiếp sử dụng đồng thời cả hai cơ chế đánh địa chỉ đồng thời, tương tự như vậy cũng có cơ chế đánh địa chỉ cơ sở/offset…Cũng có một số cơ chế tự động tăng hoặc giảm thanh ghi sử dụng trong lệnh đang thực thi nhờ vậy mà có thể giảm được độ lớn của mã chương trình đặc biệt phù hợp cho các ứng dụng Nhúng.

Tác vụ và truyền thông giữa các tác vụ

Các tác vụ (Task)

Một hệ thống thời gian thực được gọi là *"điều khiển sự kiện"* có nghĩa là hệ thống đó phải có chức năng chính là phản ứng lại các sự kiện xảy ra trong môi trường của hệ thống. Vậy thì hệ thống phản ứng lại các sự kiện như thế nào? Một giải pháp đưa ra có tên *Đa nhiệm*. Giải pháp này đã được chứng minh là một mô hình chuẩn cho các hệ thống điều khiển sự kiện và hệ thống sử dụng ngắt. Ý tưởng cơ bản của giải pháp này là chúng ta có thể phân chia một vấn đề lớn thành các nhánh nhỏ và đơn giản hơn để giải quyết. Mỗi một vấn đề con (*sub-problem*) trở thành một tác vụ - task. Mỗi một tác vụ chỉ làm một việc đơn giản. Sau đó, chúng ta giả thiết rằng các tác vụ này chạy song song với nhau. Trên thực tế, các tác vụ không bao giờ chạy song song nếu chúng ta không có một hệ thống đa vi xử lý. Trong trường hợp đang xét, các tác vụ sẽ chia sẻ một bộ vi xử lý.

Cũng giống như các chương trình khác, một tác vụ bao gồm mã lệnh để thực hiện các chức năng tác vụ phải thực hiện (do người lập trình đã thiết kế). Mã lệnh được chứa trong một hàm tương tự như hàm main() trong ngôn ngữ lập trình C. Điều làm nên sự khác biệt của tác vụ chính là ngữ cảnh (context) chứa trong ngăn xếp (stack) của nó.



Task là gì?

Mỗi một tác vụ bao gồm :

- Mã nguồn chứa các chức năng của tác vụ.
- Một ngăn xếp để chứa ngữ cảnh của tác vụ.
- Một hộp thư (mail box) (tùy chọn) để phục vụ cho việc truyền thông với các tác vụ khác.

Chú ý rằng, đôi khi (nhiều khi khá hữu dụng) ta có thể tạo ra nhiều tác vụ từ một hàm chung. Như đã nói, điều làm cho một tác vụ có thể tách biệt và khác biệt với các tác vụ

khác chính là ngăn xếp của nó. Thực tế đây chính là lập trình hướng đối tượng kiểu cổ điển. Ta có thể nghĩ rằng hàm tác vụ chính là việc định nghĩa một class. Và một tác vụ tạo ra từ hàm đó chính là một ví dụ về class.

Mặc dù có thể thấy các tác vụ là khá độc lập, nhưng về cơ bản thì chúng cũng cần phải hợp tác với nhau để thực hiện một mục đích chung đã được thiết kế sẵn cho hệ thống. Vì vậy, mỗi một tác vụ cần phải có một cơ chế truyền thông mà thông qua đó, chúng có thể kết nối, đồng bộ với các tác vụ khác. Trong trường hợp này, ta gọi cơ chế đó là Hộp thư – mail box.

Hình 7.2 miêu tả cấu trúc mã nguồn của một tác vụ. Đối số *data* dùng để tham số hóa một tác vụ. Vai trò của nó cũng giống với các đối số *argv* và *argc* trong hàm *main()* với ngôn ngữ C. Đối số này thực sự quan trọng trong trường hợp nhiều tác vụ cùng được tạo ra từ một hàm. Sự duy nhất của tác vụ được thể hiện bởi giá trị của đối số này.

```
void task(void* data)
    {
        init_task();
        while (true)
        {
        Wait for message at task mailbox();
           switch(Message.type)
           {
              case MESSAGE_TYPE_X;
                 ...
                    break;
              case Message_TYPE_Y;
                 ...
                    break;
           }
        }
    }
```

Cấu trúc thông thường của một tác vụ

Một tác vụ có thể được khởi động với một vài khởi tạo (có thể bao gồm khởi tạo đối số data). Sau đó, thông thường, tác vụ sẽ đi vào một vòng lặp không giới hạn. Tại một vài điểm trong vòng lặp, nó sẽ đợi *"Một sự kiện nào đó xảy ra",* có thể, sự kiện đó là một bản tin được gửi tới mail box, hoặc đơn giản là tràn bộ định thời. Trong khi chờ sự kiện, tác vụ sẽ không làm gì cả và không sử dụng bộ vi xử lý. Một vài tác vụ khác nếu đã sẵn sàng hoạt động hoặc đang hoạt động sẽ xử dụng bộ vi xử lý.

Khi sự kiện mà tác vụ đang chờ xảy ra, tác vụ sẽ "thức dậy" và: nhận lấy bản tin, giải mã bản tin và hoạt động theo các yêu cầu đặt sẵn dựa trên một hệ thống các yêu cầu được phân định bởi câu lệnh *switch().* Sau khi thực hiện xong yêu cầu, tác vụ lại quay trở lại trạng thái chờ sự kiện.

Có thể thấy rằng, tất cả các tá vụ đều giành phần lớn thời gian cho việc chờ một sự kiện nào đó xảy ra. Đây cũng chính là lí do để đa nhiệm hoạt động.

Truyền thông và đồng bộ giữa các tác vụ

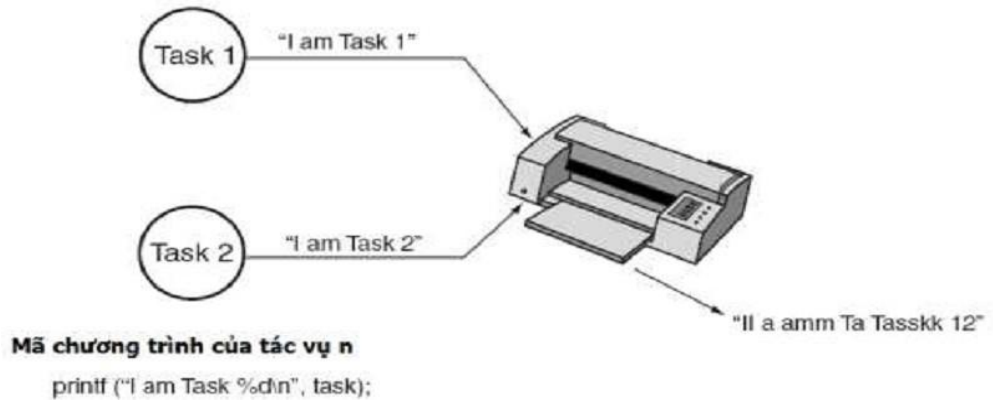
Mặc dù các tác vụ được xem như độc lập với nhau nhưng nhiệm vụ tổng quát của hệ thống yêu cầu các tác vụ phải có sự liên hệ với nhau, hợp tác với nhau. Do đó, thành phần cốt yếu của bất cứ hệ điều hành thời gian thực là tập hợp các dịch vụ truyền thông và đồng bộ.

Có một vài cơ chế đồng bộ và truyền thông hay được sử dụng, bao gồm:

- Semaphore: Sử dụng cho việc đồng bộ hóa tín hiệu và khả năng tận dụng tài nguyên.
- Monitor: Điều khiển việc truy cập vào vùng dữ liệu chia sẻ trong hoạt động của hệ thống.

Semaphore

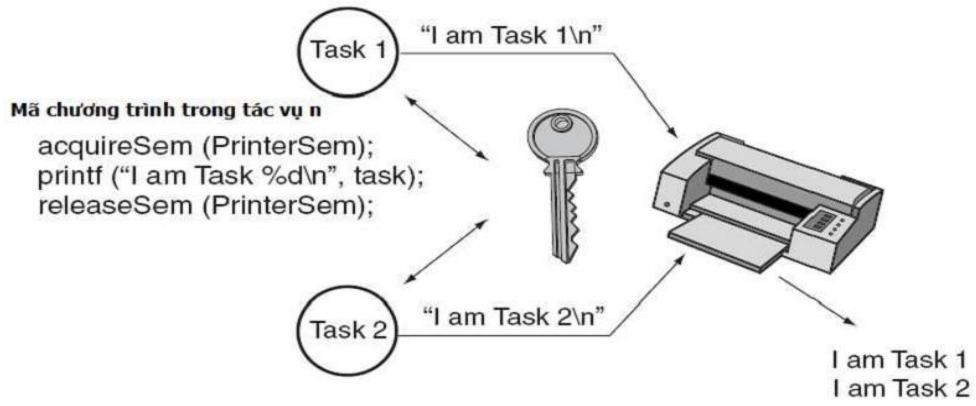
Hãy xét 2 tác vụ, mỗi tác vụ có nhiệm vụ in một bản tin có nội dung *"I am task n"* (n la số thứ tự của tác vụ) bằng một máy in chia sẻ như trong hình dưới. Nếu chúng ta không sử dụng bấ t cứ một cơ chế đồng bộ nào, kết quả có được từ máy in sẽ có thể là *"II a amm tatasskk 12".*



chia sẻ tài nguyên

Điều cần thiết ở đây là phải có một cơ chế nào đó để với cơ chế này, máy in chỉ có thê được sử dụng bởi 1 tác vụ tại một thời điểm xác định.

Semaphore hoạt động giống như một chiếc chìa khóa cho việc truy cập tới tài nguyên. Chỉ có tác vụ có chìa khóa này mới có quyền sử dụng tài nguyên. Để có thể sử dụng tài nguyên (là chiếc máy in trong trường hợp này), tác vụ cần yêu cầu (acquire) chìa khóa (semaphore) bằng cách gọi tới một dịch vụ thích hợp như trong hình 7.4. Nếu chìa khóa ở trạng thái sẵn sàng, tức là tài nguyên (máy in) hiện tại không được sử dụng bởi bất kỳ một tác vụ nào, tác vụ đó có thể được cho phép sử dụng tài nguyên. Sau khi sử dụng xong, tác vụ đó phải trả lại (*release*) semaphore cho các tác vụ khác có thể sử dụng.



Chia sẻ tài nguyên với Semaphone

Tuy nhiên, nếu máy in đang được sử dụng, tác vụ đó sẽ bị khóa cho tới khi tác vụ đang sử dụng máy in trả lại *semaphore*. Cùng một lúc có thể có nhiều tác vụ yêu cầu *semaphore* trong khi máy in đang hoạt động. Tất cả các tác vụ đó đều sẽ bị khóa. Các tác vụ bị khóa sẽ được xếp hàng theo kiểu hàng đợi theo thứ tự về mặt ưu tiên hoặc theo thứ tự thời gian mà chúng yêu cầu *semaphore* theo lệnh *acquireSem*. Cách thức sắp xếp thứ tự hàng đợi cho các tác vụ có thể được xây dựng trong *kernel* hoặc cũng có thể được cấu hình khi mà *semaphore* được tạo ra

Lệnh *acquireSem* hoạt động như sau:

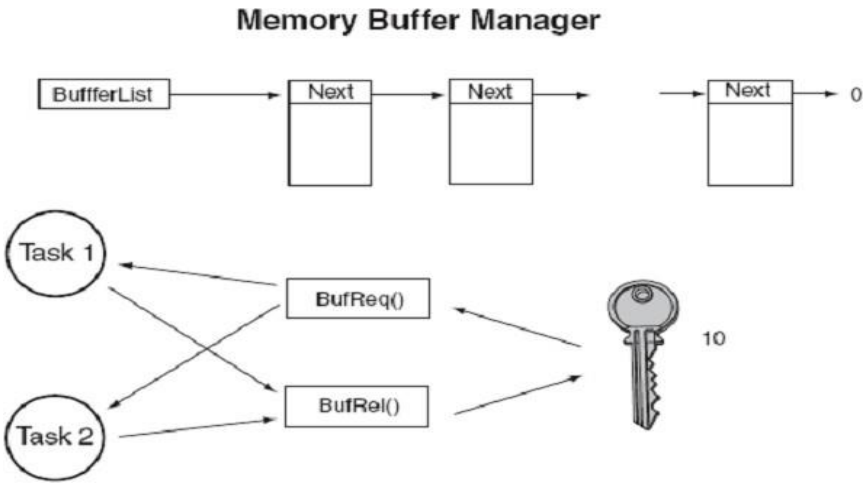
- Giảm giá trị của semaphore.
- Nếu kết quả giá trị lớn hơn hay bằng 0, tức là tài nguyên là sẵn sàng, tác vụ có thể sử dụng tài nguyên ngay lập tức. Ngược lai kết quả nhỏ hơn 0, tác vị sẽ bị khóa và chờ đến khi tác vị đang sử dụng tài nguyên sử dụng lệnh *releaseSem*.

Lệnh *releaseSem* tăng giá trị của semaphore, Nếu kết quả trả về bé hơn hoặc bằng 0, điều đó có nghĩa là có ít nhất một tác vụ đang đợi semaphore, do đó tác vụ sẽ được chuyển vào trạng thái sẵn sàng.

Trong trường hợp máy in này, semaphore sẽ được gán mặc định ban đầu là 1 trong trường hợp hệ thống chỉ có một máy in được quản lý. Trường hợp này thông thường được gọi là semaphore nhị phân (*binary semaphore*) để phân biệt với các trường hợp tổng quát hơn (*counting semaphore*), trong đó semaphore được mặc định là một số bất kỳ nguyên và không âm.

Xét một bộ định địa chỉ bộ nhớ động để quản lý bộ nhớ đệm cố định như trên hình 7.5. Ở đây chúng ta khởi tạo cho semaphore một số lượng bộ nhớ đệm đang còn trống tại thời điểm ban đầu. Khi câu lệnh *bufReq* được gọi đến, nó trước tiên dành lấy semaphore, sau đó định địa chỉ cho bộ nhớ đệm. Trong 10 lần gọi lệnh *bufReq* đầu tiên, semaphore vẫn còn không âm, điều này làm cho các tác vụ yêu cầu semaphore vẫn có thể hoạt động

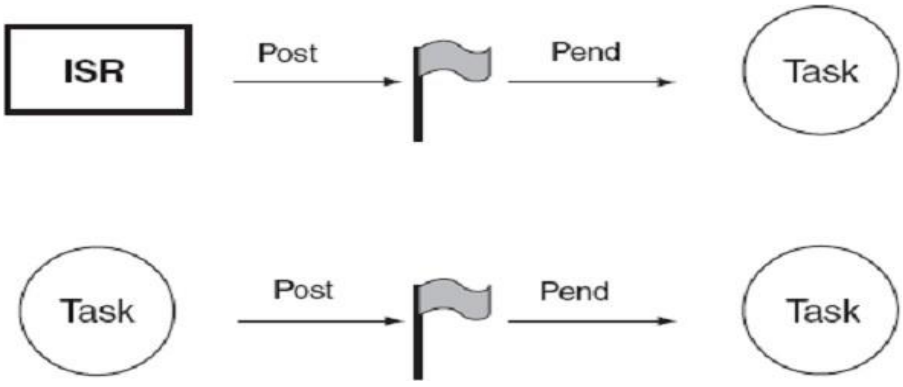
được. Đến lần thực thi lệnh *bufReq* lần thứ 11, tác vụ yêu cầu sẽ bị khóa và chờ đến khi có một tác vụ khác gọi lệnh *bufReq* để giải phóng semaphore.



Chia sẻ hệ thống đa tài nguyên

Một số kenel sử dụng cả hai loại *binary semaphore* và *couting semaphor*e vì trong một một số trường hợp, binary semaphore có hiệu quả hơn. *Binary semaphone* đôi khi còn được gọi là *mutex* có nghĩa là loại trừ lẫn nhau (*mutual exclusion*) .

Một semaphore đôi khi cũng có thể được sử dụng để tạo tín hiệu cho sự xuất hiện của một sự kiện như trong hình 7.6. Lấy vụ dụ làm thế nào mà hệ thống có thể nhận biết được sự xuất hiện của một ngắt? tác vụ cần thông tin về sự xuất hiện của ngắt sẽ treo (*pend*) semaphore lên. Chương trình con dịch vụ ngắt (*Internet sevice Routine*) sẽ phục vụ ngắt và sau đó gửi (post) semaphore lại. (chú ý rằng: thuật ngữ "*pend*" và "*post*" được sử dụng thường xuyên hơn các thuật ngữ "*acquire*" và "*release".*



Tạo tín hiệu cho sự kiện thông qua semaphore

Ở các ví dụ trên, semaphore được khởi tạo bởi một giá trị khác 0 bởi vì tài nguyên là sẵn sàng để sử dụng. Ở đây, semaphore được khởi tạo là 0. Vì vậy khi tác vụ đầu tiên treo

(pend) semaphore, nó ngay lập tức bị khóa lại - sự kiện chưa được sảy ra. Khi một ISR gửi (post) lại semaphore, tác vụ đó tiếp tục được đánh thức và tiếp tục được thực hiện .

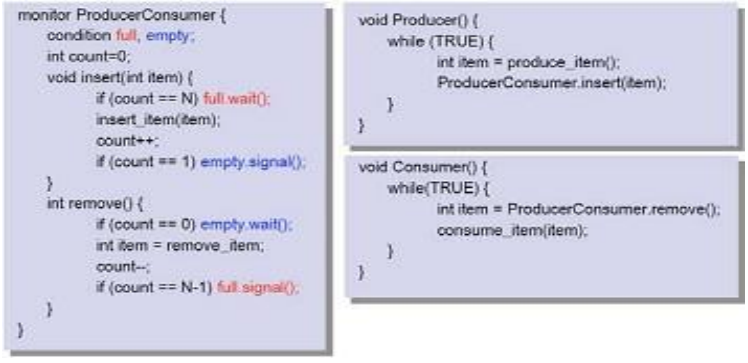
Khi semaphore được sử dụng như một khóa tài nguyên, nhiều tác vụ có thể post hoặc pend nó. Tuy nhiên trong trường hợp tạo tín hiệu hoặc đồng bộ hóa, semaphore thường được sử dụng bởi chỉ một ISR và một 1 tác vụ.

Cơ chế tương tự như trên cũng có thể được sử dụng khi một tác vụ muốn tạo tín hiệu của một sự kiện tới một tác vụ khác.

Monitor

Monitor là một ngôn ngữ lập trình được xây dựng để điều khiển việc truy nhập vào vùng dữ liệu chia sẻ trong hoạt động của hệ thống. Mã chương trình đồng bộ được bổ sung vào trong bộ biên dịch và thực thi khi chạy chương trình.

- Monitor là một modul đóng gói
- Các cấu trúc dữ liệu được chia sẻ.
- Các thủ tục hoạt động thao tác trên các cấu trúc dữ liệu chia sẻ.
- Đồng bộ các luồng thực thi đồng thời mà có thể kích hoạt các thủ tục trong hoạt động hệ thống.
- Monitor có thể bảo vệ dữ liệu khỏi sự truy nhập không có cấu trúc. Nó đảm bảo rằng các luồng truy nhập vào dữ liệu thông qua các thủ tục tương tác theo những cách hợp pháp và có kiểm soát.
- Monitor đảm bảo loại trừ xung đột
- Chỉ có một luồng có thể thực thi bất kỳ thủ tục nào tại mỗi một thời điểm (luồng trong monitor)
- Nếu có một luồng đang thực thi bên trong một *monitor* nó sẽ khoá các luồng khác muốn vào, do đó *monitor* cũng phải có một hàng đợi.



Minh họa về Monitor

Kĩ thuật lập lịch và xử lý ngắt trong thời gian thực

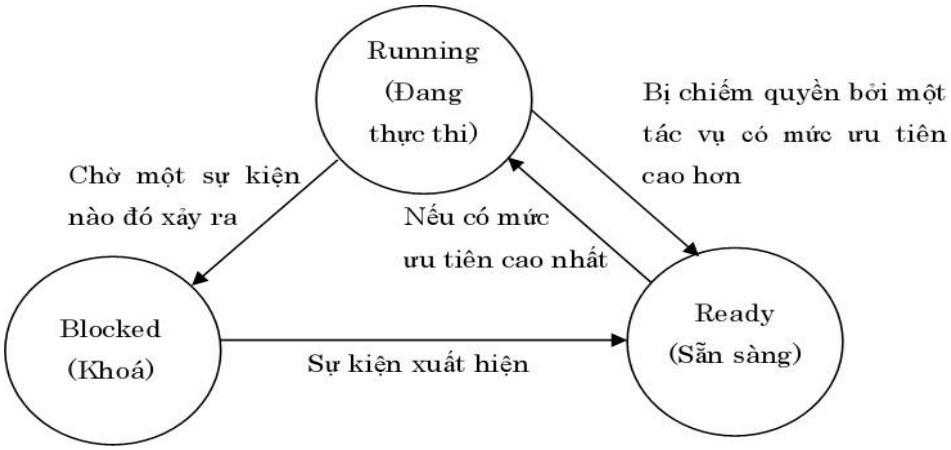
Các khái niệm

Các tác vụ (task) hoạt động dưới sự giám sát của kernel thời gian thực. Chúng bao gồm:

- Một tập hợp các dịch vụ thực hiện các công việc như đồng bộ hoá và giao tiếp truyền thông giữa các tác vụ.
- Một bộ lập lịch (scheduler) với chức năng khẳng định rằng chỉ có duy nhất tác vụ với mức ưu tiên cao nhất đang được thực thi.

Bộ lập lịch xét các tác vụ như những cái máy trạng thái (state machine). Tất cả các kernel đều có mô hình trạng thái của nó, tuy nhiên, thông thường thì các mô hình trạng thái nảy rất phức tạp. Hình 9.1 chỉ ra cho các bạn thấy một mô hình trạng thái mang tính khái niệm của tác vụ. Trong hình, ta thấy có các trạng thái:

- *Đang thực thi* (*Running*): chỉ có duy nhất một tác vụ là được nằm trong trạng thái này. Một tác vụ có thể tự động chuyển từ trạng thái Đang thực thi sang trạng thái Khoá (Blocked) bằng việc chờ đợi một sự kiện xảy ra. Trong một hệ thống có sự *chiếm quyền thực thi* (chúng ta sẽ đề cập đến nó sau), bộ lập lịch có thể bắt một tác vụ đang ở trạng thái Đang thực thi xuống trạng thái Sẵn sàng (Ready) nếu có một tác vụ với mức ưu tiên cao hơn chuyển đến trạng thái Sẵn sàng. Chúng ta gọi nó là *sựchiếm quyền thực thi* (*preemption*).
- *Sẵn sàng* (*Ready*): nếu một tác vụ đã sẵn sàng để hoạt động nhưng lại có mức ưu tiên thấp hơn tác vụ đang thực thi, tác vụ đó sẽ được chuyển đến trạng thái này và chờ. Tác vụ này sẽ được chuyển đến trạng thái Đang thực thi nếu nó trở thành tác vụ có mức ưu tiên cao nhất.
- *Khoá* (*Blocked*): một tác vụ bị khoá là tác vụ đang đợi một sự kiện nào đó xảy ra, ví dụ như một bản tin, tin nhắn được gửi đến hộp thư của tác vụ đó, hay thời gian chờ của tác vụ kết thúc….



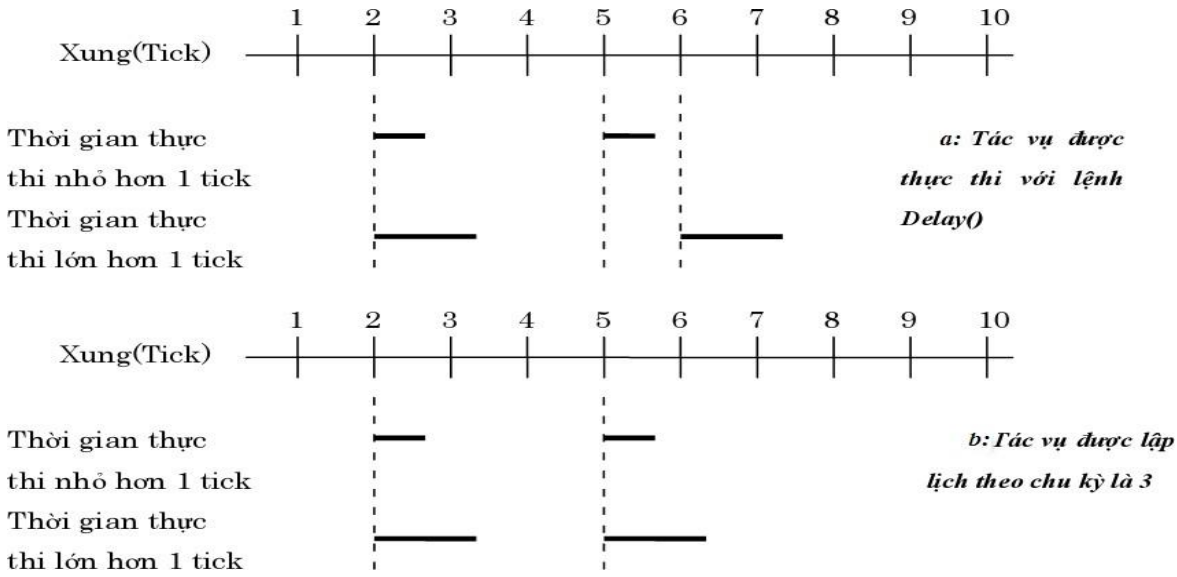
Mô hình trạng thái của tác vụ

Các phương pháp lập lịch phổ biến

Lập lịch có chu kỳ

Có rất nhiều tác vụ mà công việc của nó chỉ là thức dậy theo chu kỳ, làm một vài công việc nào đó và quay trở lại ngủ tiếp. Có một vài phương pháp để thực hiện các tác vụ kiểu này như trên hình 8.2. Trong tất cả các hệ điều hành, chúng ta đều có thể tìm thấy một lệnh gọi là hàm trễ *Delay(),* hoặc là một vài hàm có chức năng tương tự. Hàm này làm cho tác vụ bị khoá trong một thời gian xác định cho trước, thông thường thời gian này được biểu diễn bằng xung đồng hồ (clock tick). Hình *8.2a* cho ta thấy việc thực hiện một tác vụ khi ta sử dụng lệnh Delay() đối với tác vụ có tính chu kỳ đó. Trong trường hợp này, khoảng thời gian trễ là 3 clock tick. Hoạt động của hệ thống phụ thuộc vào thời gian thực thi của tác vụ. Nếu thời gian thực hiện nhỏ hơn 1 tick thì tác vụ sẽ thức dậy sau mỗi 3 tick như mong muốn. Tuy nhiên, nếu tác vụ hoạt động quá 1 tick, khi đó, sau khi tác vụ gọi lệnh Delay(), nó vẫn sẽ bị khoá trong 3 clock tick. Thế nhưng, trong ví dụ này, tác vụ thực tế là sẽ thức dậy sau mỗi 4 xung clock tick. Đó không phải là điều chúng ta mong muốn.

Một phương án khác, không phải hệ thống nào cũng có, được trình bày ở hình *8.2b*. Trong trường hợp này, bộ lập lịch sẽ đánh thức tác vụ vào đúng thời điểm thích hợp mà không quan tâm đến thời gian thực hiện tác vụ. Do đó, thay vì dùng hàm Delay(), một tác vụ có tính chu kỳ sẽ gọi hàm WaitTilNext(). Hàm này sẽ khóa tác vụ cho tới phiên thực hiện kế tiếp.



Các tác vụ có tính chu kì

Lập lịch không theo chu kỳ

Một số tác vụ phải phản ứng lại các sự kiện xảy ra ngẫu nhiên tại các thời điểm khác nhau. Một sự kiện có thể là việc một gói dữ liệu từ trên mạng được gửi đến nơi, việc một cái công tắc đóng lại để chỉ ra là bể nước đã đầy hoặc cũng có thể là sự kết thúc việc convert một tín hiệu tương tự sang số của bộ ADC và đang cần được đọc. Thông thường, các sự kiện không đồng bộ này được giao tiếp với máy tính thông qua các ngắt. Chương trình con dịch vụ ngắt phải có cách nào đó để kết nối sự xuất hiện của ngắt với tác vụ chịu trách nhiệm xử lý sự kiện.

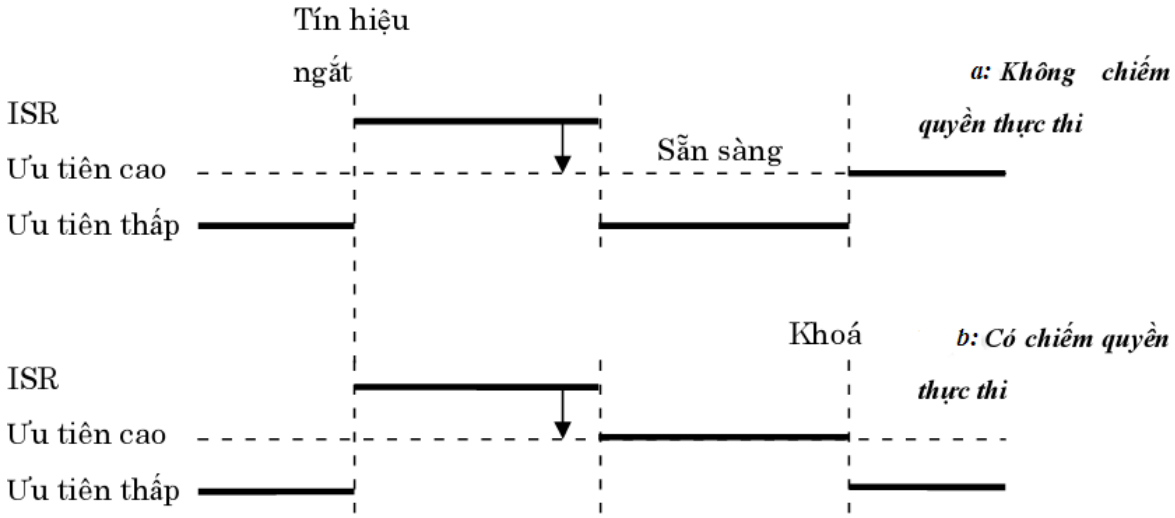
Lập lịch theo kiểu chiếm quyền thực thi và lập lịch không có chiếm quyền thực thi

Có 2 phương thức cơ bản cho việc lập lịch một tác vụ: *chiếm quyền thực thi* và *không chiếm quyền thực thi*. Xét 2 tác vụ: tác vụ 1 có mức ưu tiên thấp hơn đang thực hiện và tác vụ 2 có mức ưu tiên cao hơn đang bị khoá để chờ một sự kiện xảy ra, sự kiện này được thông báo bởi một tín hiệu ngắt. Hình 8.3a cho thấy những gì xảy ra trong hệ thống không có tính chiếm quyền ưu tiên. Chương trình con dịch vụ ngắt ISR làm cho tác vụ 2 với mức ưu tiên cao hơn chuyển từ trạng thái Khoá sang trạng thái Sẵn sàng. Tuy nhiên, đến khi ISR được thực hiện xong thì tác vụ 1 với mức ưu tiên thấp hơn vẫn sẽ được tiếp tục thực thi tại điểm nó bị ngắt. Sau đó, khi tác vụ 1 bị khoá để chờ sự kiện thì tác vụ 2 mới được chuyển sang trạng thái thực thi.

Hình *8.3 b* ứng với trường hợp của hệ thống có tính chiếm quyền ưu tiên. Điểm khác biệt ở đây là bộ lập lịch được gọi đến ở cuối chương trình con dịch vụ ngắt. Bộ lập lịch xác định tác vụ có mức ưu tiên cao đang ở trạng thái Sẵn sàng và chuyển nó lên trạng

thái thực thi. Do đó, tác vụ với mức ưu tiên thấp đã bị chiếm quyền thực thi. Một hệ thống không có tính chiếm quyền thực thi muốn rằng tất cả các tác vụ phải là "những công dân tốt" bằng cách tự nguyện trao trả bộ xử lý cho các tác vụ khác để chắc chắn một điều: các tác vụ đều có cơ hội để sử dụng bộ xử lý. Các thế hệ Windows trước kia đều là dạng này. Linux thì khác, nó là một hệ điều hành có tính chiếm quyền thực thi mặc dù các bản Linux chuẩn không quan tâm đến vấn đề thời gian thực bởi trong một thời gian dài, vấn đề chiếm quyền thực thi không được đề cập.

Các hệ thống có tính chiếm quyền thực thi cung cấp cho ta nhiều hơn thời gian phản ứng có thể dự đoán được bởi vì tác vụ có mức ưu tiên cao sẽ được xử lý ngay lập tức. Đây chính là điểm cốt lõi của thời gian thực: khả năng đảm bảo một thời gian lớn nhất để phản ứng lại một sự kiện. Trong hệ thống không chiếm quyền thực thi, chẳng có gì để đảm bảo thời gian một tác vụ nhường lại bộ xử lý cho các tác vụ khác. Mặt khác, trong một hệ thống có chiếm quyền ưu tiên, vấn đề tranh chấp tài nguyên hệ thống cũng đáng được quan tâm cẩn thận.



Lập lịch: Có và không có chiếm quyền thực thi

Hai phương án khác được tận dụng để xử lý các tác vụ có cùng mức ưu tiên. Trong phương thức lập lịch kiểu *vòng lặp robin*, một tác vụ sẽ được thực thi đến khi nào nó bị khoá (block) để chờ một sự kiện xuất hiện hoặc cũng có khi nó tình nguyện nhường (yield) bộ xử lý lại. Sự khác biệt giữa khoá và nhường là ở chỗ: trong trường hợp thứ 2, tác vụ sẽ quay trở lại trạng thái Sẵn sàng (ready). Xét trường hợp trong danh sách Sẵn sàng có 3 tác vụ thứ tự lần lượt là A, B, C. Các tác vụ này có cùng mức ưu tiên như nhau. Tác vụ A đứng đầu danh sách và được chuyển đến trạng thái Thực thi. Khi tác vụ A nhường (yield) bộ xử lý, tác vụ B trở thành trạng thái thực thi và danh sách Sẵn sàng sẽ như sau:

B C A

Khi B nhường, C sẽ chuyển trạng thái và danh sách sẽ chuyển thành:

B C A

Như vậy, tất cả các tác vụ sẽ hoạt động thành một vòng tròn, chúng hoạt động theo kiểu nhường nhau. Các tác vụ có mức ưu tiên thấp hơn trong trạng thái Sẵn sàng sẽ không bao giờ được thực hiện cho đến khi tất cả các tác vụ trên bị khoá.

Nhát cắt thời gian là một biến thể của vòng lặp robin. Trong đó, nó quy định mỗi tác vụ sẽ nhận được một lượng thời gian nhất định hay còn gọi là nhát cắt thời gian. Việc làm này bảo vệ các tác vụ khỏi trường hợp chiếm dụng bộ xử lý quá lâu. Do đó, một tác vụ sẽ chạy cho đến khi nó bị khoá, nó tình nguyện nhường hay quá hạn về thời gian cho phép. Tùy thuộc vào hoàn cảnh và yêu cầu, các tác vụ sẽ có lượng thời gian cho phép bằng nhau hoặc khác nhau. Xét trên khía cạnh nào đó, vòng lặp robin chỉ là một dạng khác của vòng lặp polling.

Kỹ thuật lập lịch

- FCFS

Trong cơ chế lập lịch đến trước được phụ vụ trước thì các quá tình được xử lý theo thứ tự mà nó xuất hiện yêu cầu và cho đến khi hoàn thành. Cơ chế lập lịch này thuộc loại không ngắt được và có ưu điểm là dễ dàng thực thi. Tuy nhiên, nó không phù hợp cho các hệ thống mà hỗ trợ nhiều người sử dụng vì có một sự biến đổi lớn về thời gian trung bình mà một quá trình hay tác vụ phải chờ đợi để được xử lý. Hơn nữa do việc xử lý không ngắt được nên có hiện tượng chiếm hữu độc quyền bộ xử lý trong thời gian dài và có thể gây ra sự trễ bất thường trong quá trình thực hiện của các tác vụ phải chờ đợi khác.

- Shortest Job First - SJF

Trong cơ chế lập lịch này tác vụ có thời gian thực thi ngắn nhất sẽ có quyền ưu tiên cao nhất và sẽ được phục vụ trước. Vấn đề chính gặp phải trong cơ chế lập lịch này là không biết trước được thời gian thực thi của các tác vụ tham gia trong chương trình và thông thường phải áp dụng cơ chế tiên đoán và đánh giá dựa vào kinh nghiệm về các tác vụ thực thi trong hệ thống. Điều này chắc chắn rất khó để luôn đảm bảo được độ chính xác. Cơ chế lập lịch này có thể áp dụng cho cả loại ngắt được và không ngắt được.

- Rate monotonic (RM):

Phương pháp lập lịch RM có lẽ hiện này là thuật toán được biết tới nhiều nhất áp dụng cho các tác vụ hay quá trình độc lập. Phương pháp này dựa trên một số giả thiết sau:

(1) Tất cả các tác vụ tham gia hệ thống phải có deadline kiểu chu kỳ

(2) Tất cả các tác vụ độc lập với nhau

(3) Thời gian thực hiện của các tác vụ biết trước và không đổi

(4) Thời gian chuyển đổi ngữ cảnh thực hiện là rất nhỏ và có thể bỏ qua

Thuật toán RM được thực thi theo nguyên lý gán mức ưu tiên cho các tác vụ dựa trên chu kỳ của chúng. Tác vụ nào có chu kỳ nhỏ thì sẽ có được gán mức ưu tiên cao. Theo nguyên lý này với các tác vụ chu kỳ không thay đổi thì RM sẽ là phương pháp lập lịch cho phép ngắt và mức ưu tiên cố định. Tuy nhiên RM hỗ trợ yêu cầu hệ thống không tốt.

- Earliest deadline first (EDF)

Như đúng tên gọi của phương pháp, thuật toán lập lich theo phương pháp này sử dụng deadline của tác vụ hay như điều kiện ưu tiên để xử lý điều phối hoạt động. Tác vụ có deadline gần nhất sẽ có mức ưu tiên cao nhất và các tác vụ có deadline xa nhất sẽ nhận mức ưu tiên thấp nhất. Ưu điểm nổi bật của phương pháp này là giới hạn có thể lập lịch đáp ứng được 100% cho tất cả các tập tác vụ. Hơn nữa mức ưu tiên gán cho mỗi tác vụ trong quá trình hoạt động là động vì vậy chu kỳ của tác vụ có thể thay đổi bất kỳ lúc nào theo thời gian. EDF có thể được áp dụng cho các tập tác vụ chu kỳ và cũng có thể mở rộng để đáp ứng cho các trường hợp các deadline thay đổi khác nhau theo chu kỳ.

Vấn đề chính của thuật toán lập lich EDF là không thể đảm bảo được tác vụ nào trong hệ thống có thể không được thực thi trong tình huống quá độ hệ thống bị quá tải. Trong nhiều trường hợp mặc dù mức độ sử dụng trung bình nhỏ hơn 100% những vẫn có thể trong một tình huống nào đó vẫn vượt qua khả năng đáp ứng của hệ thống tức là sẽ có tác vụ không được đảm bảo thực thi đúng. Trong những trường hợp như vậy cần phải điều khiển để biết tác vụ nào bị lỗi không thực hiện thành công hoặc tác vụ nào được thực hiện thành công trong quá trình quá độ.

- Minimum Laxity first (MLF)

Cơ chế lập lịch này sẽ ưu tiên tác vụ nào còn ít thời gian còn lại để thực hiện nhất trước khi nó phải kết thúc để đảm bảo yêu cầu thực thi đúng. Đây được xem là cơ chế lập lịch gán quyền ưu tiên động và dễ đạt được sự tối ưu về hiệu suất thực hiện và sự công bằng trong hệ thống.

- Round Robin

Đây là một cơ chế lập lịch phân bổ đều đặn, ngắt được và đơn giản. Mỗi một tác vụ được xử lý/phục vụ trong một khoảng thời gian nhất định và lặp lại theo một chu trình xuyên suốt toàn bộ các tác vụ tham gia trong hệ thống. Khoảng thời gian phục vụ cho mỗi tác vụ trong quá trình là một sự thoả hiệp giữa thời gian thực hiện của các tác vụ và thời gian thực hiện một chu trình. Có thể chọn khoảng thời gian đó rất nhỏ và đôi lúc chúng ta không nhận được ra rằng đang có sự phân bổ thực hiện trong hệ thống. Tuy nhiên nếu thời gian đó quá nhỏ có thể làm mất tính hiệu quả thực hiện toàn hệ thống vì cần nhiều thời gian trong việc chuyển đổi ngữ cảnh cho mỗi tác vụ sau mỗi chu trình thực hiện.

Xử lý ngắt

Một hệ thống thời gian thực được gọi là *"điều khiển sự kiện"* có nghĩa là hệ thống đó phải có chức năng chính là phản ứng lại các sự kiện xảy ra trong môi trường của hệ thống. Vậy thì hệ thống phản ứng lại các sự kiện như thế nào?. Hiện nay có hai phương pháp tiếp cận vấn đề này. Phương pháp đầu tiên là *Polling* hay *Vòng lặp Polling* và phương pháp thứ 2 là xử lý ngắt (Interrup).

Polling

```
int main (void)
{
sys_init();
while (TRUE)
{
        if (event_1)
            service_event_1();
        if (event_2)
            service_event_2();
            .
            .
        if (event_n)
            service_event_n();
    }
}
```

Vòng lặp Polling

Hãy xem đoạn code trong hình 8.4. Chương trình được bắt đầu bằng một vài cài đặt ban đầu cho hệ thống rồi truy cập vào trong một vòng lặp vô hạn, trong đó, các sự kiện mà hệ thống có thể phản ứng lại được kiểm tra. Khi có một sự kiện xảy ra, dịch vụ phản ứng lại sự kiện đó được kích hoạt.

Tiến trình thực hiện của vòng lặp trên tỏ ra khá đơn giản và thích hợp với những hệ thống nhỏ không đòi hỏi quá gắt gao về mặt thời gian. Tuy nhiên, cũng có một số vấn đề cần bàn đến:

- Thời gian phản ứng lại sự kiện phụ thuộc rất lớn vào vị trí mà chương trình đang thực hiện trong vòng lặp. Lấy ví dụ: Nếu sự kiện *event_1* xảy ra ngay

trước câu lệnh *if(event_1)* thì thời gian phản ứng là rất ngắn. Nhưng nếu sự kiện *event_1* xảy ra ngay sau khi câu lệnh kiểm tra đó, chương trình lúc này phải quét toàn bộ vòng lặp và quay trở về điểm đầu và thực hiện dịch vụ của sự kiện *event_1*.

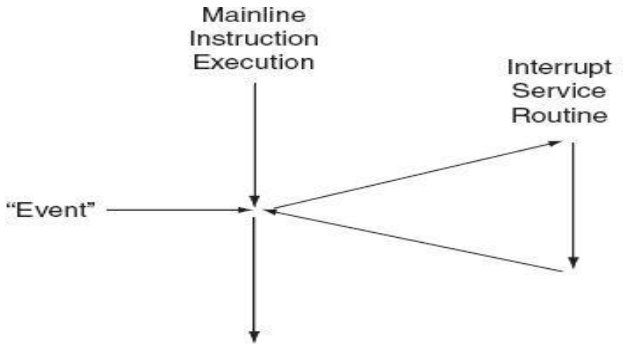
- Và cũng là một hệ quả tất yếu, thời gian phản ứng cũng là một hàm của số lượng sự kiện được kích hoạt tại một thời điểm và sau đó là thời gian thực hiện các dịch vụ trong một lần quét vòng lặp của chương trình.
- Tất cả các sự kiện được chương trình đối xử một cách bình đẳng và không có sự ưu tiên nào cả.
- Khi có một đặc tính mới, do đó là dịch vụ mới, được thêm vào chương trình, thời gian phản ứng lại dài thêm ra.

Interrupt (ngắt)

Phương pháp tiếp cận thứ 2 là *ngắt* (*Interrupt*). Rất hữu dụng và cũng gây không ít khó khăn cho người lập trình. Ý tưởng của ngắt: sự xuất hiện của một sự kiện có thể "ngắt" tiến trình thực hiện của chương trình, "nhồi" thêm và thực hiện một tiến trình khác vào như hình 8.5. Khi tiến trình nhồi thêm được thực hiện xong, chương trình chính lại được thực hiện tiếp từ thời điểm bị ngắt. Tiến trình của sự kiện ngắt được thực hiện ngay lập tức mà không phải quan tâm đến chương trình chính. Những tiến trình như thế người ta gọi là "*chương trình con dịch vụ ngắt*" (*Interrupt Service Routine*) viết tắt *ISR*.

Các thế hệ vi xử lý hiện nay thường thực hiện 3 loại ngắt khác nhau:

- Câu lệnh INT, hay nhiều khi còn được nhắc đến với cái tên là TRAP (bẫy). Nó như một câu lệnh để gọi một chương trình con đặc biệt. Chúng ta sẽ đề cập đến nó sau.
- Các trường hợp đặc biệt của bộ xử lý. Các điều kiện lỗi như lỗi chia 0, lỗi truy cập bất hợp pháp vào bộ nhớ có thể được điều khiển thông qua cơ chế ngắt.



Ngắt

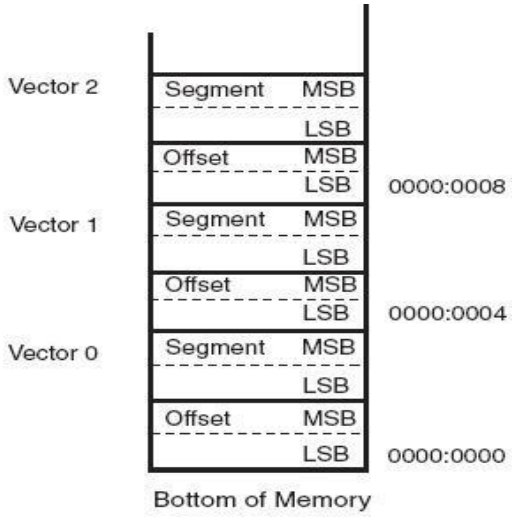
Hai loại ngắt kể trên đồng bộ với việc thực hiện lệnh. Trong đó: INT chính là một câu lệnh và các lỗi đặc biệt của bộ xử lý chính là kết quả trực tiếp của việc thực hiện lệnh.

Loại ngắt thứ 3 được tạo ra bởi sự kiện xảy ra bên ngoài bộ xử lý. Loại ngắt này được tạo ra bởi các I/O phần cứng và xảy ra không đồng bộ với việc thực hiện lệnh.

Các ngắt ngoài không đồng bộ:

- Làm tối đa hoá hiệu suất và thông lượng của hệ thống máy tính
- Gây ra phần lớn các lỗi và rắc rối cho người lập trình

Hầu hết các bộ xử lý đều sử dụng lược đồ ngắt giống nhau. Hình 8.6 chỉ ra kiến trúc ngắt của Intel x86. 1kilo byte (KB) đầu tiên của bộ nhớ được giành cho *bảng véctơ ngắt* (*Interrupt Vector Table*). Mỗi một véctơ có 4 byte thể hiện địa chỉ (segment và offset) của chương trình con dịch vụ ngắt. Các véctơ này mang những ý nghĩa, chức năng khác nhau và được định nghĩa bởi kiến trúc của bộ xử lý. Ví dụ: véctơ 0 là lỗi chia 0, véctơ 3 là *breakpoint* (lệnh ngắt INT 1byte).



Ngắt- Bảng véctơ ngắt

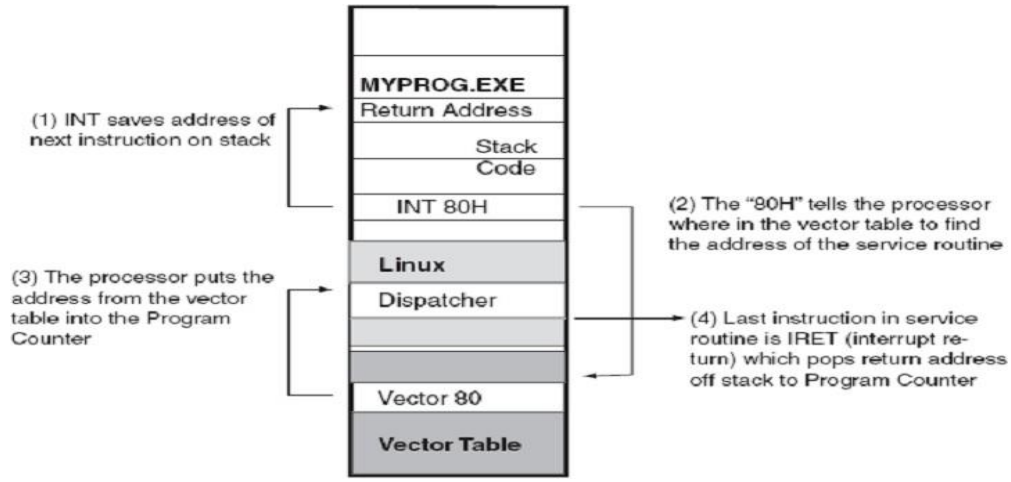
Một số véctơ được dành cho các ngắt ngoài. Trong PC, các véctơ 8 đến 15 và 0x70 đến 0x77 được dành cho phần cứng.

Tất cả các véctơ được truy cập thông qua lệnh ngắt INT 2byte, trong đó, byte thứ 2 chỉ ra số thứ tự của véctơ (ngắt). Phần mềm hệ thống thường thiết lập các quy ước liên quan đến nhiều véctơ này. Ví dụ: PC BIOS sử dụng một số ngắt cho các dịch vụ phần cứng và LINUX sử dụng ngắt INT 0x80 để gọi dịch vụ của kernel.

Sau đây là một ví dụ về việc sử dụng ngắt INT 0x80 của Linux:

- Bộ xử lý lưu lại giá trị hiện thời của thanh ghi bộ đếm chương trình Program Counter (PC) và Code Segment (CS) vào ngăn xếp stack cùng với từ điều khiển trạng thái bộ xử lý Proccesor Status Word (PSW).

- Byte thứ 2 trong câu lệnh INT là một chỉ số trong bảng véctơ ngắt để từ đó tìm được địa chỉ của chương trình con dịch vụ ngắt (ISR). Bộ xử lý nạp địa chỉ này vào thanh ghi PC và CS và việc thực hiện chương trình con được thực hiện từ điểm này.

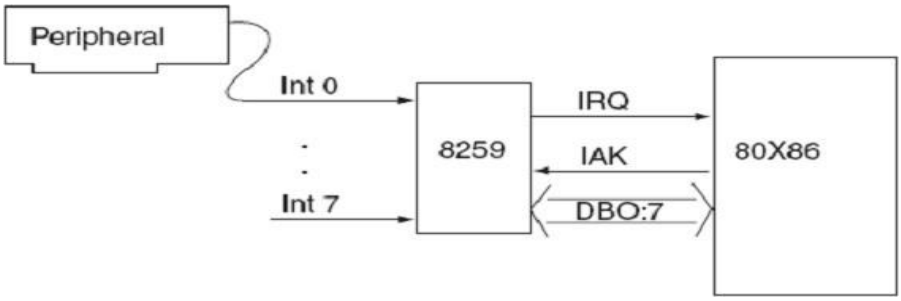


Ngắt và hoạt động của ngắt

- Kết thúc của ISR là câu lệnh IRET (Interrupt Return). Nó giải phóng PC và CS để nạp lại giá trị của chương trình chính và thực hiện tiếp lệnh tiếp theo sau lệnh INT.

Lệnh INT cũng tương tự như lệnh gọi chương trình con CALL nhưng có đôi chút khác biệt: trong khi địa chỉ đích của lệnh CALL được nhúng vào trong câu lệnh đó thì với INT, ta không cần quan tâm đến địa chỉ của ISR. Địa chỉ của nó nằm trong bảng véctơ ngắt. Đây là một điểm thuận lợi cho việc truyền thông giữa chương trình được biên dịch và chương trình được tải, ví dụ như chương trình ứng dụng và hệ điều hành.

Các ngắt ngoài có cách thức thực hiện như thể hiện trong hình 8.8. Một thiết bị bên ngoài đưa ra một *"yêu cầu ngắt" Interrupt Request* (*IRQ*). Khi bộ xử lý phản ứng lại bằng một xác nhận "*chấp nhận ngắt*" *Interrupt Acknowledge* (*IAK*), thiết bị đó sẽ gửi số thứ tự của véctơ ngắt lên bus dữ liệu. Bộ xử lý sau đó sẽ thiết lập một lệnh ngắt INT với chỉ số véctơ ngắt đã được cung cấp.

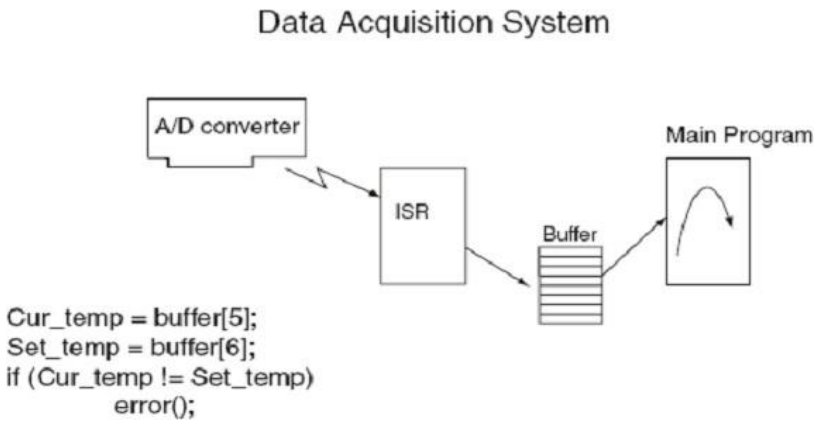


Ngắt cứng

Hình 8.8: Ngắt cứng

Ngắt có thể được kích hoạt hoặc bị vô hiệu hoá. Ở cấp độ của bộ xử lý, ngắt có thể được kích hoạt hoặc vô hiệu hoá thông qua câu lệnh STI và CLI. Các ngắt có thể được kích hoạt hoặc vô hiệu một cách có chọn lọc ở cả bộ điều khiển ngắt 8259 hay ở chính thiết bị đó. Trên thực tế, *việc kích hoạt và vô hiệu ngắt chính làđiểm mấu chốtđể thiết kế và thực thi một phần mềm thời gian thực*

Cũng không có gì đáng ngạc nhiên khi nói rằng ngắt không đồng bộ có những vấn đề đáng bàn của nó. Để ý một ứng dụng thu thập dữ liệu dựa trên bộ A/D đa kênh như trên hình 8.9. Cứ mỗi khi bộ chuyển đổi A/D thu thập một tập hợp dữ liệu trên các kênh, nó ngắt bộ xử lý. Chương trình con dịch vụ ngắt đọc dữ liệu và cất vào bộ nhớ đệm, nơi mà chương trình khác (còn gọi là chương trình nền) sẽ tiếp tục xử lý.



Ví dụ về ngắt

Hoạt động điều khiển ngắt cho phép chúng ta phản ứng lại A/D một cách nhanh chóng trong khi bộ nhớ đệm tách chương trình nền khỏi nguồn dữ liệu, ví dụ: chương trình nền không cần quan tâm đến dữ liệu được từ đâu mà có được. Bây giờ hãy xem đến đoạn mã lệnh được ghi trong hình 8.9. Giả thiết chỉ là thí nghiệm, chúng ta cung cấp một tín hiệu biến đổi liên tục vào cả kênh 5 và 6. Đồng thời, giả thiết rằng chương trình sẽ không bị "fail" khi đang thực hiện đo tín hiệu đồng nhất.

Trong thực tế, chương trình như đã viết chắc chắn sẽ bị "fail" bởi vì một ngắt có thể xảy ra trong khi cập nhật biến Cur_temp và cập nhật biến Set_temp với kết quả là giá trị của biến Cur_temp được cập nhật từ tập hợp dữ liệu cũ trước đó, còn giá trị của biến Set_temp được cập nhật từ tập hợp dữ liệu hiện thời. Như vậy, khi tín hiệu đầu vào thay đổi theo thời gian và các tập hợp dữ liệu được tách rời nhau ở các thời gian xác định, giá trị các biến sẽ khác nhau và do đó, chương trình sẽ "fail".

Đây chính là bản chất của vấn đề lập trình thời gian thực. *Cần phải quản lý các ngắt khôngđồng bộđể chúng không xảy ra vào những thờiđiểm không thích hợp.*

Có một giải pháp, dù không hay cho lắm, để giải quyết vấn đề này. Ta có thể dùng một lệnh vô hiệu hoá ngắt (CLI) trước khi cập nhật biến *Cur_temp* và kích hoạt ngắt bằng lệnh STI sau khi cập nhật biến *Set_temp*. Việc làm này giúp các ngắt tránh khỏi phiền phức của việc cập nhật liên tục như đã đề cập. Có vẻ như chúng ta đã sáng suốt khi sử dụng các lệnh CLI và STI như một chìa khoá cho một giải pháp đúng đắn, nhưng nếu chỉ đơn giản là rải các lệnh CLI và STI trong code của chương trình thì cũng chẳng khác gì việc sử dụng các lệnh “go to” và các biến toàn cục.

Thiết kế Hệ thống nhúng

Quy trình phát triển của một hệ thống nhúng

Quá trình phát triển của một hệ thống nhúng được thực hiện theo chu trình sau:

- (1) Problem specification
- (2) Tool/chip selection
- (3) Software plan
- (4) Device plan
- (5) Code/debug
- (6) Test
- (7) Integrate

Mô hình hóa sự kiện và tác vụ

Phương pháp mô hình Petrinet

Năm 1962 *Carl Adam Petri* đã công bố phương pháp mô hình hình hoạ tác vụ hay quá trình theo sự phụ thuộc nhân quả đã được phổ cập rộng rãi và được biết tới như ngày này với tên gọi là mạng Petri.

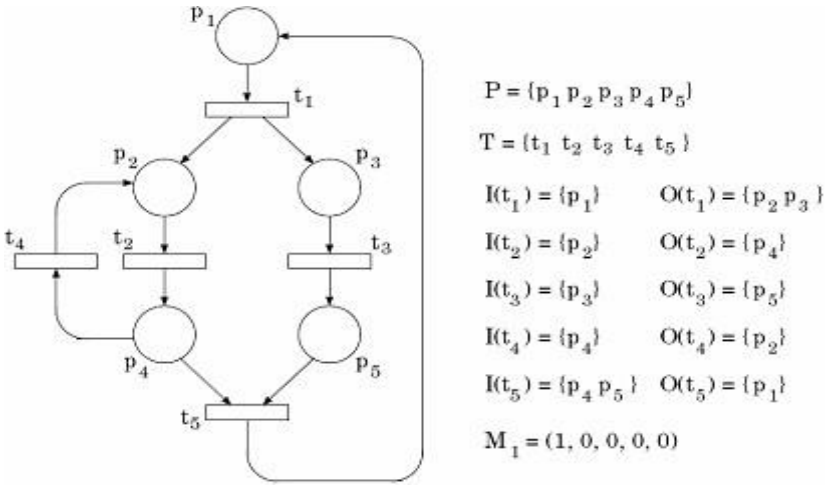
Mạng Petri được sử dụng phổ biến để biểu diễn mô hình và phân tích các hệ thống có sự cạnh tranh trong quá trình hoạt động. Một hệ thống có thể hiểu là một tổ hợp của nhiều thành phần và mỗi thành phần thì đều có các thuộc tính. Các thuộc tính đó có thể thay đổi và được đặc trưng bởi các biến trạng thái. Một chuỗi các trạng thái sẽ mô tả quá trình động của một hệ thống.

Mạng Petri thực sự là một giải pháp mô tả hệ thống động với các sự kiện rời rạc tác động làm thay đổi trạng thái của các đối tượng trong hệ thống theo từng điều kiện cụ thể trạng thái của hệ thống.

Mạng Petri được thiết lập dựa trên 3 thành phần chính: (1) Các điều kiện, (2) các sự kiện, và (3) quan hệ luồng. Các điều kiện có thể là thoả mãn hoặc không thoả mãn. Các sự kiện là có thể xảy ra hoặc không. Và quan hệ luồng mô tả điều kiện của hệ trước khi sự kiện xảy ra.

Các điều kiện đòi hỏi phải thoả mãn để một sự kiện xảy ra hoặc chuyển trạng thái thực hiện thì được gọi là điều kiện trước (*precondition*). Các điều kiện mà được thoả mãn khi một sự kiện nào đó xảy ra thì được gọi là điều kiện sau (*postcondition*).

Quy ước biểu diễn mô hình Petrinet



Ví dụ về mô hình mạng Petri

Trong qui ước biểu diễn hình hoạ thì mạng Petri sử dụng các vòng tròn để biểu diễn các điều kiện, các hộp để biểu diễn các sự kiện, và mũi tên biểu diễn quan hệ luồng. Một ví dụ minh hoạ về mạng Petri được mô tả trong Hình 13.1, trong đó:

- P = {p**1** , p**2** ,..., p**np** } là tập gồm np vị trí được biểu diễn trong mô hình (được mô tả bởi các vòng tròn).
- T = {t**1** , t**2** ,..., t**nt** } là tập gồm nt chuyển đổi trong tập chuyển đổi biểu diễn trong mô hình (được mô tả bởi các hình chữ nhật).
- I biểu diễn quan hệ đi vào chuyển đổi và được ký hiệu bởi đường mũi tên theo hướng từ các vị trí tới các chuyển đổi.
- O biểu diễn quan hệ đi ra khỏi chuyển đổi và được ký hiệu bởi các đường mũi tên theo hướng từ các chuyển đổi tới các vị trí.
- M = {m**1** , m**2** ,...m**np** } là dấu trạng thái của các chuyển đổi trong hệ thống. Các giá trị m**i** là số các thẻ bài (được ký hiệu như các chấm tròn đen) chứa bên trong các vị trí p**i** trong tập dấu M .

Hệ thống động có thể được mô tả bởi mạng Petri nhờ sự chuyển dịch các thẻ bài trong các vị trí của hệ thống mô hình và tuân thủ theo luật sau:

- Một chuyển đổi được phép thực thi nếu tất cả các vị trí đi vào chuyển đổi đó chứa ít nhất một thẻ bài.
- Khi một chuyển đổi đã được thực thi xong (hoàn thành) thì một thẻ bài sẽ bị loại ra khỏi vị trí đi vào chuyển đổi đó đồng thời bổ sung thêm một thẻ bài vào các vị trí đầu ra tương ứng của chuyển đổi đó.

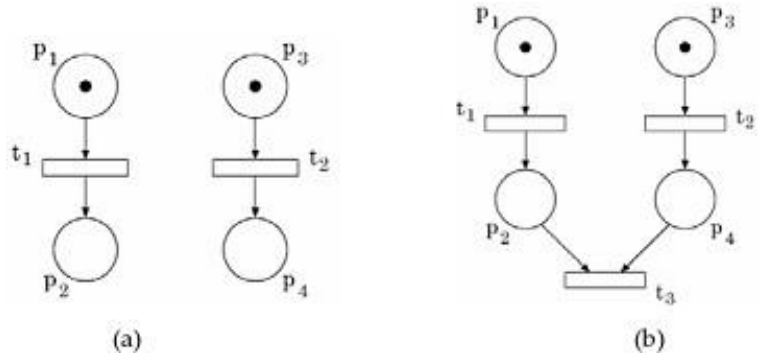
Các trạng thái động của hệ thống được mô tả bởi tập *R(M)* đánh dấu bởi các dấu trong tập M. Trong ví dụ trên có 5 phần tử dấu trong tập R lần lượt là M 1 , M 2 , M 3 , M 4 , M 5. Tương ứng lần lượt như sau:

- M 1 = (1, 0, 0, 0, 0)
- M 2 = (0,1,1, 0, 0)
- M 3 = (0,1, 0, 0,1)
- M 4 = (0, 0, 0,1,1)
- M 5 = (0, 0,1,1, 0)

Mô tả các tình huống hoạt động cơ bản với Petrinet

- Song song và đồng bộ:

Trong mô hình PN mô tả như trong Hình 9.2 (a), các chuyển đổi t**1** và t**2** được phép thực hiện đồng thời; hoạt động của chúng không ảnh hưởng đến nhau. Các hoạt động được mô hình bởi hai chuyển đổi thực hiện song song. Trong hệ thống dự phòng với độ tin cậy cao, mô hình này được sử dụng để biểu diễn hai thành phần C**1** và C**2** song song để đảm bảo hoạt động dự phòng; trong trường hợp này các vị trí p**1** và p**3** biểu diễn điều kiện làm việc, các vị trí p**2** và p**4** biểu diễn điều kiện lỗi, t**1** và t**2** là các sự kiện lỗi trong các tác vụ C**1** và C**2** một cách tương ứng.



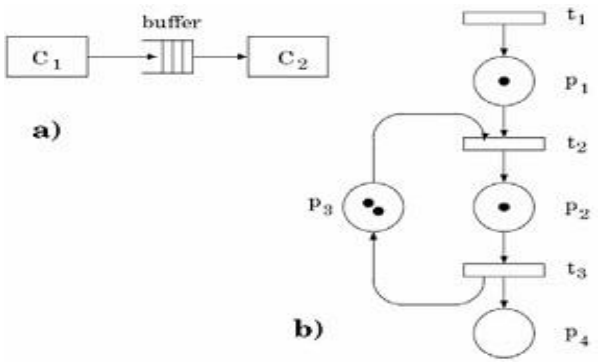
Mô hình Petrinet 2 hoạt động song song: Độc lập (a) và đồng bộ (b)

Trong hoạt động song song, các tác vụ hoàn toàn độc lập, tuy nhiên nếu các sự kiện đó cần phải kết thúc và là điều kiện để cho một chuyển đổi khác thì hoạt động đồng bộ có thể được thực hiện nhờ bổ sung một chuyển đổi t**3** như mô tả trong Hình 9.2 (b). Khi đó chuyển đổi t**3** cần thẻ bài đồng thời của cả p**2** và p**4**.

- Chia sẻ đồng bộ:

Một yếu tố đặc trưng trong hoạt động của hệ thống phân tán là thường phải chia sẻ một số tài nguyên hữu hạn. Sự thiếu thốn về tài nguyên làm hạn chế hoạt động của hệ thống trong quá trình xử lý thậm chí làm tắc nghẽn hệ thống. Việc mô hình và phân tích các

hệ thống có hiện tượng tắc nghẽn là một tác vụ khó khăn trong hầu hết các quá trình mô hình có thể gặp phải.

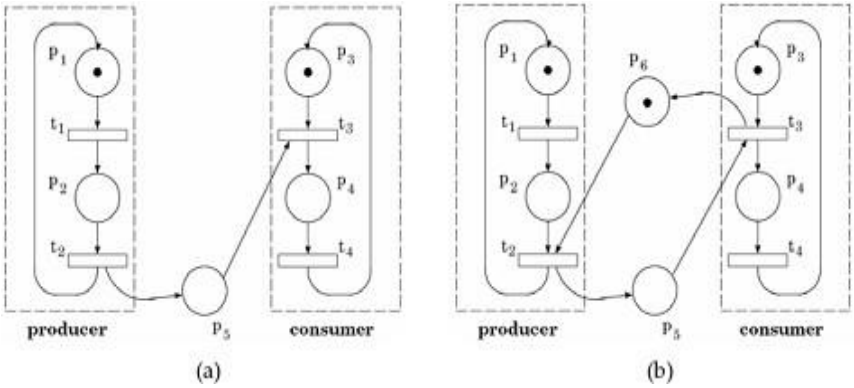


Hoạt động của bộ đệm với dung lượng hữu hạn

Để minh hoạ tình huống này, biểu diễn hoạt động của bộ đệm với dung lượng hữu hạn được mô tả bởi PN trong Hình 9.3. Vị trí p**3** mô hình số các vị trí bộ đệm còn trống và vị trí p**2** mô hình số vị trí đã được điền đầy; chú ý rằng tổng các thẻ bài chứa trong các vị trí p**2** và p**3** luôn là hằng số (trong ví dụ này là 3). Chuyển đổi t**2** mô hình quá trình điền đầy một vị trí bộ đệm và hoàn thành nếu có ít nhất một vị trí bộ đệm còn trống cùng với thẻ bài chứa trong vị trí p**1** và p**3**. Chuyển đổi t**3** được phép thực hiện nếu có ít nhất một vị trí bộ đệm đã được điền đầy. Khi hoàn thành chuyển đổi t**3**, một thẻ bài sẽ được chuyển từ vị trí p**2** sang vị trí p**3.**

- Tuần tự:

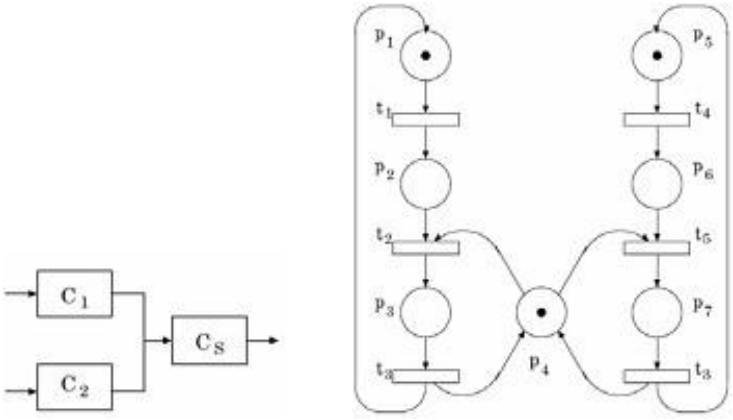
Hoạt động tuần tự sẽ được mô tả và minh hoạ bởi hoạt động của bộ tạo và bộ sử dụng thông qua một bộ đệm. Bộ tạo sẽ sinh ra các đối tượng để đưa vào trong một bộ đệm và sẽ được lấy ra bởi bộ sử dụng. Quá trình sử dụng sẽ phải được thực hiện một cách tuần tự theo quá trình tạo ra đối tượng. Mô hình cho hoạt động này được diễn tả bởi PN như trong Hình 9.4 (a). Thẻ bài chứa trong vị trí p**1** có nghĩa là bộ tạo đã sẵn sàng thực hiện. Khi các chuyển đổi t**1** và t**2** hoàn thành thì một đối tượng được tạo ra (một thẻ bài tương ứng cũng sẽ được chuyển vào trong bộ đệm mô hình bởi vị trí p**5**) và bộ tạo lại sẵn sàng trở lại. Nếu bộ sử dụng có nhu cầu tiêu thụ (được mô hình bởi thẻ bài chứa trong vị trí p**3**) và đang có ít nhất một đối tượng trong bộ đệm thì một thẻ bài chứa trong vị trí p**5** sẽ được lấy đi và chuyển đổi t**3** sẽ hoàn thành.



Hoạt động tạo và sử dụng với bộ đệmvô hạn (a) và hữu hạn(b)

Trong cách mô tả trong Hình 9.4 (a) thì việc tạo và sử dụng được thực hiện thông qua một bộ đệm với giả thiết là có dung lượng vô hạn. Trong thực tế thì các bộ đệm là hữu hạn, để mô tả hoạt động với bộ đệm loại này Hình 9.4 (b) được sử dụng. Vị trí p**6** mô hình các vị trí bộ đệm còn trống và vị trí p**5** mô hình các vị trí bộ đệm đã được điền đầy. Tổng số lượng các thẻ bài chứa trong các vị trí p**5** và p**6** phải luôn là hằng số. Nếu một thẻ bài được gán cho vị trí p**5** trong dấu khởi tạo thì bộ tạo sẽ không thể tạo thêm đối tượng chừng nào bộ sử dụng vẫn chưa tiêu thụ đối tượng trong bộ đệm.

Loại trừ xung đột



Hoạt động loại trừ của hai tác vụ song song chia sẻ tài nguyên

Hai tác vụ C1 và C2 được phép làm việc song song và cùng chia sẻ tài nguyên CS, nhưng không được truy nhập vào tài nguyên đồng thời. Giản đồ PN cho hoạt động này được mô tả như trong Hình 10.5. Các vị trí p1 và p5 biểu diễn các tác vụ C1 và C2 làm việc độc lập; vị trí p2 và p6 biểu diễn các yêu cầu của các tác vụ C1 và C2 một cách tương ứng khi muốn truy nhập vào tài nguyên chia sẻ CS; p3 và p7 biểu diễn CS đang bị chiếm dụng bởi các tác vụ C1 và C2 một cách tương ứng. Vị trí p4 mô tả quyết định xem tác vụ nào có thể truy nhập tài nguyên Cs và tránh các vị trí p3 và p7 bị đánh dấu đồng thời. Thực tế khi p2 và p6 được đánh dấu thì các chuyển đổi t2 và t5 xung đột. Việc hoàn thành một trong hai tác vụ sẽ khoá/cấm lẫn nhau. Việc hoàn thành chuyển đổi

t3 hoặc t6 sẽ mô hình việc giải phóng nguồn tài nguyên chung (chuyển thẻ bài trở lại vị trí p4) và trở về điều kiện làm việc bình thường.

Ngôn ngữ mô tả phần cứng (VHDL)

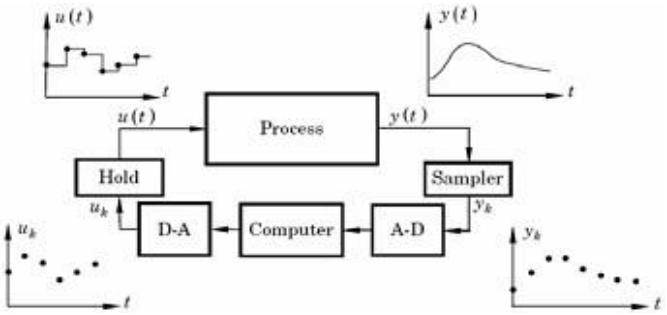
VHDL (*Very High Speed Integrated Circuit Hardware Description Lanuage*) là một ngôn ngữ chung để mô tả các thiết kế phần cứng ở mức phần tử logic cơ bản cấu thành nên hệ thống và đã được phát triển bởi tổ chức quốc phòng Mỹ. Mục đích chính là để thuận tiện cho việc trao đổi dữ liệu thiết kế phần cứng theo một định dạng chuẩn mà mọi người có thể hiểu và thông dịch, tạo điều kiện thuận lợi trong việc phối hợp hay hợp tác trong các dự án thiết kế. Đặc biệt nó rất thuận tiện trong việc chuyển đổi hay tổng hợp biên dịch thành một dạng ngôn ngữ thực thi phần cứng thực. Điều này rất khó thực hiện bằng các ngôn ngữ bậc cao như C nhưng với VHDL điều này chính là ưu điểm nổi bật và là thế mạnh trong việc mô hình hoá hệ thống, mô tả một cách chi tiết các phần tử cứng cấu thành tham gia trong hệ thống.

VHDL là một chuẩn IEEE (Std - 1076) đã được sự hỗ trợ bởi rất nhiều nhà cung cấp phát triển phần cứng. Ứng dụng một cách chuyên nghiệp ngôn ngữ này là phục vụ cho việc mô tả các mạch ASICs phức hợp, chế tạo thực thi các mạch FPGA...

Ngôn ngữ VHDL có thể đọc hiểu khá dễ dàng với cấu trúc cú pháp rõ ràng gần giống như ngôn ngữ Visual Basic và Pascal. Nó có thể phát huy được thế mạnh về cú pháp để định nghĩa xây dựng kiểu dữ liệu mới và hỗ trợ cho việc lập trình theo nhóm. Với xu thế hiện nay các nhóm phát triển có thể thực thi với điều kiện cách xa nhau về khoảng cách địa lý, vì vậy việc phối hợp và thiết kế theo nhóm là rất cần thiết.

Thiết kế các phần mềm điều khiển

Thiết kế phần mềm điều khiển



Hệ thống điều khiển số

Để thực thi một bộ điều khiển số trên thiết bị vật lý thực phải đòi hỏi xét xem bộ điều khiển với mô hình hàm truyền đã cho có thể hiện thực hóa được không. Điều kiện phải xét thực ra là để đảm bảo rằng không có đầu ra nào của hệ thống lại xuất hiện trước khi có tín hiệu vào. Hay nói cách khác hệ thống xây dựng phải tuân thủ tính nhân quả.

Nếu khai triển hàm truyền của bộ điều khiển số được mô tả ở dạng tổng quát:

$$G_R(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_m z^{-m}}{a_0 + a_1 z^{-1} + \cdots + a_n z^{-n}}$$

thành chuỗi lũy thừa theo z thì nó phải không được phép chứa bất kỳ phần tử nào chứa lũy thừa dương của z. Hay nói cách khác là bộ điều khiển được mô tả như trên phải có bậc ≤ 0 tức là bậc của tử số phải nhỏ hơn hoặc bằng bậc của mẫu số ($n \geq m$). Sau khi đã thiết kế được bộ điều khiển số thì việc còn lại là lập trình và nạp vào các bộ điều khiển vật lý khả trình. Thực chất quá trình này là thực thi hàm truyền của bộ điều khiển số bằng lập trình số trên các bộ điều khiển vật lý đã có. Ở đây chúng ta sẽ chủ yếu quan tâm đến việc triển khai để chuẩn bị cho bước lập trình các hàm truyền của bộ điều khiển số. Xuất phát từ mô tả hàm truyền dạng tổng quát của bộ điều khiển số:

$$G_R(z) = \frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_m z^{-m}}{a_0 + a_1 z^{-1} + \cdots + a_n z^{-n}}$$

trong đó, $a0 \neq 0$ nếu $b0 \neq 0$; m và n là các số nguyên dương.

Có thể triển khai để thực thi một hàm truyền của bộ điều khiển số theo 3 cách như sau:

Triển khai lập trình số trực tiếp

Để triển khai lập theo phương pháp lập trình trực tiếp thì hàm truyền bộ điều khiển đã cho biểu diễn trong miền z phải được chuyển đổi về dạng hàm truyền rời rạc:

$$a_0 u^*(t) + \sum_{k=1}^{n} a_k u^*(t-kT) = \sum_{k=0}^{m} b_k e^*(t-kT)$$

Từ đẳng thức trên dễ dàng tính ra được giá trị của đầu ra u * (t) của bộ điều khiển số đã cho theo các giá trị hiện tại và quá khứ của đầu vào e* (t) cũng như các giá trị quá khứ của chính nó

$$u^*(t) = \frac{1}{a_0}\sum_{k=0}^{m} b_k e^*(t-kT) - \frac{1}{a_0}\sum_{k=1}^{n} a_k u^*(t-kT)$$

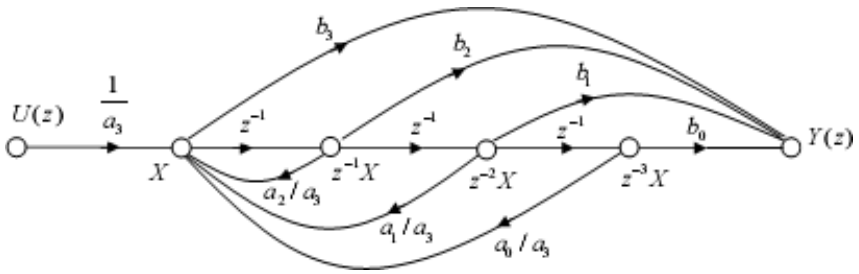
Để thực hiện bộ điều khiển này yêu cầu phải lưu trữ các giá trị quá khứ của đầu vào và đầu ra của bộ điều khiển. Với bộ điều khiển đã cho yêu cầu phải có n + m giá trị cần phải lưu trữ hay nói cách khác cần phải có n + m phần tử lưu trữ.

Một phương pháp khác để triển khai lập trình trực tiếp là sử dụng cơ chế tách trực tiếp đầu vào và đầu ra của bộ điều khiển theo một biến trung gian X(z). Không mất tính tổng quát nếu chúng ta nhân cả tử và mẫu của hàm truyền bộ điều khiển số đã cho với một biến X(z). Từ đó rút ra được hàm truyền của đầu vào E(z) theo X(z) và hàm truyền của đầu ra U(z) theo X(z). Phương pháp này thực hiện như sau:

$$U(z) = \frac{1}{a_0}(b_0 + b_1 z^{-1} + \cdots + b_m z^{-m})X(z)$$

$$X(z) = \frac{1}{a_0}E(z) - \frac{1}{a_0}(a_1 z^{-1} + a_2 z^{-2} + \cdots + a_n z^{-n})X(z)$$

Theo phương pháp này yêu cầu số phần tử lưu trữ chính bằng giá trị n, bằng bậc của đa thức mẫu số trong hàm truyền bộ điều khiển số đã cho. Từ 2 đẳng thức trên ta cũng dễ dàng xây dựng được giản đồ trạng thái mô tả hàm truyền của bộ điều khiển số (giả thiết m = n = 3).



Giản đồ trạng thái của hệ thống số

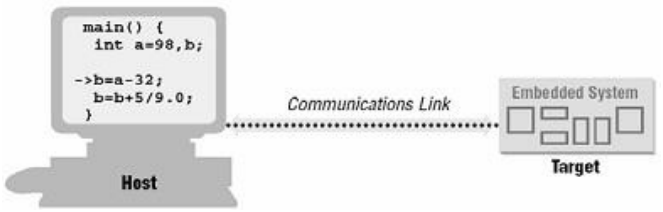
- Triển khai lập trình số ghép tầng

Cách triển khai này yêu cầu chuyển đổi bộ điều khiển về dạng tích của các hàm truyền đơn giản để có thể dễ dàng thực hiện bằng các chương trình đơn giản. Hay nói cách khác bộ điều khiển số đã cho là kết quả ghép tầng của nhiều bộ điều khiển nhỏ.

- Triển khai lập trình số song song

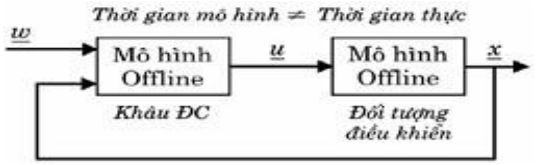
Bộ điều khiển đã cho sẽ được tách ra thành tổng của các bộ điều khiển đơn giản và có thể thực hiện lập trình song song cho các bộ điều khiển đó.

Một số phương pháp phát triển phần mềm nhúng



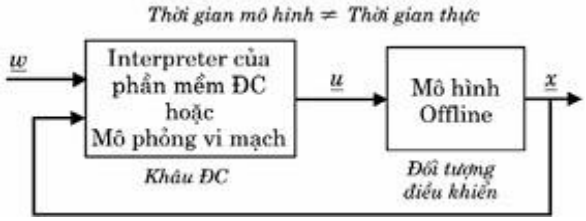
Trong quá trình phát triển, phần mềm cần phải được thử nghiệm với đối tượng điều khiển. Tuỳ thuộc vào từng môi trường phát triển chúng ta có thể tiến hành theo một số các phương pháp sau.

- Mô hình Offline



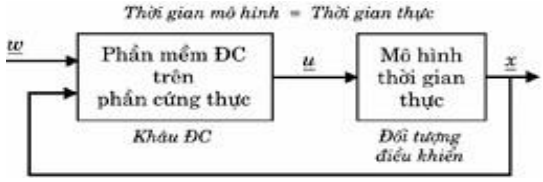
Trong hệ thống phát triển này nền phần cứng nhúng đích được mô phỏng bằng mô hình chạy trên PC và đối tượng điều khiển cũng là mô hình mô phỏng chạy trên PC. Vì vậy quá trình phát triển thực chất là quá trình chạy mô phỏng hệ thống được thực hiện hoàn toàn trên PC. Với hệ thống này không thể thử nghiệm cho các sự kiện đáp ứng thời gian thực vì thời gian của mô phỏng khác với thời gian diễn biến thực của hệ thống.

- Hệ thống phát triển (Software in the loop)



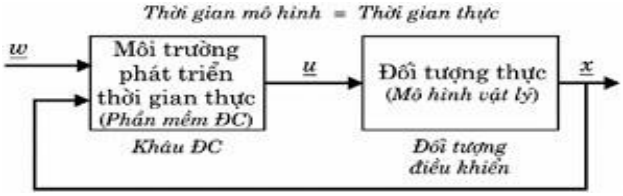
Hệ thống này mô phỏng nền phần cứng thực trên PC cho đáp ứng hành vi giống như với vi mạch cứng thực và mô hình đối tượng được mô hình thực thi trên PC. Loại hệ thống này cũng tương tự như hệ thống mô phỏng offline tuy nhiên có ưu điểm hơn vì khả năng mô phỏng hành vi và đáp ứng của vi mạch nhúng chính xác hơn và trung thực hơn. Và cũng có một nhược điểm là không thử nghiệm được bài toán thời gian thực.

- Mô phỏng thời gian thực



Hệ thống này sử dụng nền phần cứng nhúng đích thực nhưng đối tượng thì chỉ là mô hình thời gian thực không phải đối tượng thực. Ưu điểm là khá mềm dẻo và thay đổi cấu hình đơn giản trong quá trình phát triển để thử nghiệm với các hành vi khác nhau của đối tượng. Rút ngắt và đơn giản hóa công việc xây dựng đối tượng.

- Mô hình phát triển thực



Hệ thống này sử dụng nền phần cứng nhúng đích thực với đối tượng thực. Tuy nhiên có sự hỗ trợ của công cụ phát triển để có thể cài đặt và thử nghiệm trực tiếp trên nền phần cứng thực. Đây là một dạng mô hình cho kết quả trung thực và chính xác nhất trong các dạng hệ thống phát triển nêu trên. Tuy nhiên các nền phần cứng này thường được phát triển và hỗ trợ bởi các nhà cung cấp để có thể tương thích với công cụ phần mềm kèm theo.