Interruption

ปัจจัยภายใน:

Flow Control เปลี่ยนค่าเปลี่ยนลำดับได้
ไปเลี่ยน register ที่ชื่อว่า Program Counter (PC)

ตอนโดน interrupt จะเก็บค่า PC ไว้ใน stack ใน main memory เมื่อทำ sub program / function เสร็จแล้ว หยุด interrupt ก็จะ เอาค่า PC มาแล้วกลับไปทำงานเดิมต่อ

ปัจจัยภายนอก (this is real interrupt):
Hardware จะมีสายสัญญาณ Interrupt ต่อผ่าน Control Unit ของ CPU อยู่ ผ่าน Interrupt Service Routine (ISR) หรือ Interrupt Handler เมื่อทำเสร็จแล้วก็กลับมา

ตัวอย่าง Interrupt:

- Program
  - overflow or division by zero
- Timer
  - ทำงานหลาย Program สลับไปมา
- I/O Controller
  - สั่งผ่าน I/O
- Hardware Failure
  - ex. Memory Parity Error

Multiple Interrupt

Multiple Interrupt Types

Sequential --> Interrupt after another Interrupt Completed
Nested --> Interrupt ซ้อน Interrupt

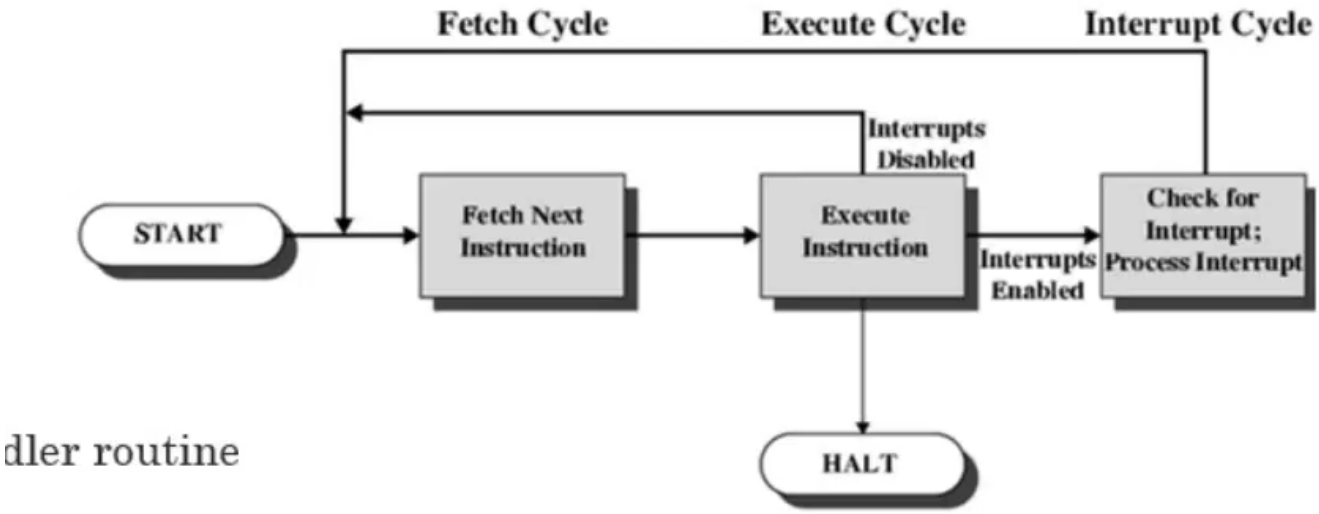
Best Practice for Multiple Interrupt

Disable Interrupt: ทำอันที่หนึ่งอยู่ มี Interrupt ซ้อนเข้ามาให้รอก่อน เมื่อทำ 1 เสร็จค่อย 2 ต่อ (Sequential)
Define Priorities: Low Priority Interrupt สามาสรถถูก Interrupt ได้ด้วย Higher Priority Interrupt

Instruction Cycle

from Fetch --> Execute to
Fetch, Execute and Interrupt

Interrupt จะทำงานหลัง Execute เสมอ (ท้ายคำสั่ง)



dler routine

d program

Instruction Set Architecture

Instruction Set --> collection of Instruction

มีการกำหนด Pattern ของรหัสคำสั่ง

Assembler --> แปลภาษา Assembly to Machine Code

Addressing Mode

เอาจตัวเลขไปทำงานกับคำสั่ง

ต้องใช้ Mode ไหน

Fundamental Modes

- Immediate
  - ตัวเลขเป็นส่วนนึงของรหัส
  - Opcode + Operand N ตัว
  - ex. ADD 5, 5 is operand เลย
  - เอาตัวเลขมาเลยไม่ต้อง fetch

    - fast, limited range
- Direct
    - ไปเอาค่าจากที่เก็บใน address นั้นโดยตรง
    - ex. ADD A, ไปเอา operand จากตำแหน่ง A
- Indirect
    - ชี้ไป address ที่ชี้ไปหา ค่า อีกทีหนึง (pointer)
    - ex. ADD (A), sometimes ADD ((((A)))) and so on
    - Multiple Memory access to find operand
    - slow

Alternative Modes

- Register
    - เรียกค่าจาก Register
    - very fast and very limited space
    - no memory access
- Register Indirect
    - Register ไปเรียกค่าจาก Memory อีกทีหนึ่ง
- Displacement (Indexed)
    - Direct Addressing + Immediate
    - Ex. ADD R1 + 10 = 2 + 10 = 12
- Relative Addressing
    - current location of PC value
    - ex. current PC pointer + 10 = 20 + 10 = 30
- Base Register
    - Base Register + Immediate (same as displacement)
- Index Addressing
    - Index Register + Immediate (same as displacement)
    - ex. IR + 10 = 2 + 10 = 12
- Stack Addressing
    - use Stack (Push, Pop)
    - ADD(SP) --> pop top 2 items and then ADD

Register Model

User Visible Register

- General Purpose
- Data, Address

- Condition Code --> บอกสถานะการทำงาน เกิด overflow เกิด error ไรไหม

Control and Status Register

- PC, IR, MAR, MBR

Other Register Model

- Depend on CPU Design
- May have register pointing to Process control blocks, Interrupt Vectors
- Special Registers

Instruction Format

Instructions Element

- Operation Code (Op-code) - Must Have
  - Command / What to do
- Associated Data (Operand)
  - can be to 0 to N
  - Normally Specific (Source of Data, Destination of Data, Next Instruction Reference)
  - Addressing Mode
  - Data Location --> CPU, Main Memory (or Virtual memory or Cache), I/O Device, Coded in instruction
- Source Operand (To this), Result Operand (Put answer here), Next instruction Reference (Next)

Instruction Representation

- Machine Code (pattern) --> binary
- human consumption --> Assembly Language / Code
- ex. ADD A, B
- can easily map Opcode, Operand into binary

Instruction Code

- Low Level
  - Machine Code
  - Assembly Code
- Intermediate / High Level
  - Human uses

- C/C++, Script-based Language, Other Language

Instruction Types

Depend on Opcode
Working with CPU Register

Data Processing --> use ALU / Operation Unit
Data Storage --> Store Number use Main Memory
Data Movement (I/O) --> Move/Transfer data, use I/O
Program Flow Control --> Change Sequence of Program, Location of Next Instruction

Types of Operand

Address
Number
Character
Logical Data --> bits or flag

Byte Order

Big Endian --> น้อยไปมาก
Little Endian --> มากไปน้อย

Types of Computer Architecture

RISC --> Reduced Instruction Set Computer (Ex. ARM) --> แต่ละคำสั่งมีขนาดเท่ากัน แต่ทำได้ไม่เยอะ, ประหยัดไฟ
CISC --> Complex Instruction Set Computer (Ex. Intel x86 64) --> คำสั่งยาวไม่เท่ากัน, หนึ่งคำสั่งทำได้หลายอย่าง, จำนวนน้อยกว่า