Computer Organization

COMP2120

Qi Zhao

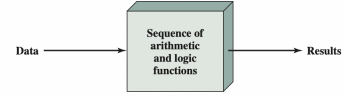
February 7, 2024

Instruction Execution Cycle

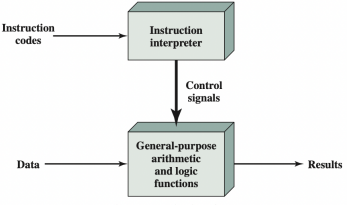
Computer Components

Key design concepts of von Neumann architecture

- Data and instructions are stored in a single read-write memory.
- The contents of memory are addressable by location (address), without regard to the type of the data or instruction.
- Execution occurs in a sequential fashion.



(a) Programming in hardware



(b) Programming in software

Figure 3.1 Hardware and Software Approaches

Top-level view of Computer Components

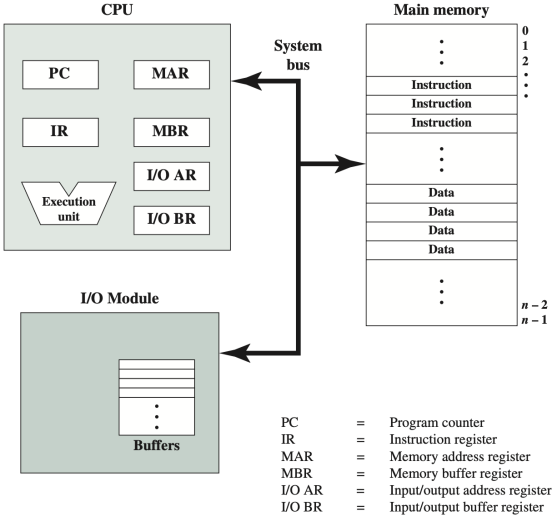


Figure 3.2 Computer Components: Top-Level View

Some basic information

- **Byte**: 8 bits, MB, million Bytes
- **Word**: the unit of organization of memory, e.g., 32 bit, 64 bit.
- **Register**: a small amount of fast storage, quickly accessible location available to a computer's processor, usually word-sized
- **Word addressing**: addresses of memory on a computer uniquely identify words of memory.
- **Byte addressing**: each byte has an address

Storage inside CPU

- **PC - Program Counter**
  holds the address of the instruction to be fetched next.
- **IR - Instruction Register**
  contains the current instruction.
- **MAR - Memory Address Register**
  specifies the **address** in memory for the next read or write. The MAR is directly connected to the external address bus to the memory.
- **MBR - Memory Buffer Register**
  (also called Memory Data Register, MDR); contains the **data** returned from the external memory, or the data to be written to the external memory.
- The data from memory during instruction fetch cycle will be fed to IR, while all other data from the memory will be fed to MBR.

Storage inside CPU

- **I/O AR - I/O Address Register**
  hold the I/O Address.
- **I/O BR - I/O Buffer Register**
  hold the **data** to/from I/O.
- You may have one or more temporary registers inside the CPU as well.
- Data is transferred via the system bus. The source register put the data on the system bus, while the destination register grab the data from the bus.
- At any one time, only one data transfer can be performed on each system bus. (i.e. multiple system bus required for parallel data transfer)

Instruction Format

- **Operation code, opcode**
  specifies the operation to be performed (ADD, LOAD)
- **Source operand reference** operands that are inputs for the operation, may involve one or more source operands.
- **Result operand reference** or destination operand. The operation may produce a result.
- **Next instruction reference** this tells the processor where to fetch the next instruction

Instruction Format

- **Immediate**: The value of the operand is contained in a field in the instruction being executed.
- **Memory**: memory address
- **Processor register**: the name or number of the desired register. CPU has a number of general purpose registers, e.g., R_1, R_2.
- **I/O device**: specify the I/O module and device

Example Instruction Format

- Instruction executed by the CPU is in 32-bit binary format
- Each operation has up to 3 operands, 2 source operands and 1 destination operand.

Op code	src operand 1	src operand 2	dst operand
1 byte=8 bits	1 byte	1 byte	1 byte

- For example, let operation code 00000000 represents an `ADD` instruction.
- Consider operation: `ADD R2, R3, R1`

$$R1 \longleftarrow R2 + R3$$

add `R2` with `R3` and put the result into R1, represented by 32-bit,

```
00000000 00000010 00000011 00000001 = 0x 00 02 03 01
```

- `ADD R4,R1,R4  0x00040104; ADD R1,R2,R1  0x00010201`

Other examples

- Also applies to other arithmetic/logic operations, such as `ADD(0)`, `SUB(1)`, `AND(2)`, `OR(3)` and `NOT(4)`, as well as `MOV(5)`, with different operation codes, representing different operations.
- for `MOV` operation, moving from source operand 1 to destination operand, and there is no source operand 2, that field will be set to `0x00`.
- `ADD/SUB` can also apply to Memory, Register, constant data.
- Reading from memory (`LD` instruction), source is memory, and destination is register.

```
LD MEM, R1 :loading from MEM (in memory) to Reg 1
```

- Writing to memory (ST instruction), source is register and destination is memory.

```
ST R1, MEM:storing content of Reg 1 to MEM
```

Other examples

- Addressing Modes: immediate, direct, indirect, register, register indirect, ...
- Number of addresses: three, two, one, zero
- If the memory size is larger than 2^8 units, how to represent the address in an instruction?
- Two-word (three-word) instructions

Other examples

- `LD` and `ST` will be a two-word instruction.
- Example: `LD A, R1; R1`$\longleftarrow$ `A, A` is in memory
  Load instruction

Op code (Load)	oooooooo	Addressing mode	dst operand
1 byte	1 byte	1 byte	1 byte

- `ST R1, A; R1`$\longleftarrow$ `A, A` is in memory
  Store instruction

Op code (Load)	Source operand	Addressing mode	oooooooo
1 byte	1 byte	1 byte	1 byte

- Only one addressing mode is used, where the target address is given by the word following the LOAD or STORE instruction. Specified as `11111111 (0xFF)`
- More addressing modes will be discussed in the chapter on instruction set.

Load and Store Instruction

- Assuming the operation code of `LD` is `0x06` and `ST` is `0x07`.
- the two-word instruction: `0600ff04  0000003c` represents a `LD` instruction, reading from memory location `0000003c`, and put in `Reg 4`

Op code (Load)	No src operand	Addressing model	dst operand
00000110	00000000	11111111	00000100

- Similary, the two-word instruction: `0704ff00 00000048` represents a `ST` instruction, reading from `Reg 4`, and store in memory location `00000048`.

Branch Instruction

- Need to change the program flow based on some condition (`if-else`).
- also called a jump instruction, one of the operands indicts the address of the next instruction to be excecuted.
- Most often, a conditional branch instruction, is made only if a certain condition is met, e.g. result of the previous arithemtic/logical operation.
- The system has a one-bit flag for each condition, e.g. zero flag: if result is zero, then branch; result is non-zero, nothing happens.
- There may be other flags, such as negative flag, overflow flag.
- The type of branch is specified in 1 bit or a few bits, called condition code.

Branch Instruction

Op code (Branch)	Condition Code (cc)	Mem Location	oooooooo
Instruction	Condition Code (cc)	Meaning
BR	0x00	always goto (unconditional)
BZ	0x01	branch if zero flag is true
BNZ	0x02	branch if zero flag is not true

- Similar to LD and ST instruction, the target address is given by the word following the instruction (use 0xFF)
- The operation code of Branch is 0x08
- The two-word instruction 0802ff00 0000001c is a branch instruction, condition code 02, is a Branch-if-not-zero, target address is 0000001c, i.e. Branch to address 1c if the previous operation gives a non-zero result, (BNZ)

Example Assembly Language Program

	Assembly Language	Address	Content	
	LD P0, R4	0000:	0600ff04	0000003c
	LD P1, R1	0008:	0600ff01	00000040
	MOV R1, R2	0010:	05010002	
	LD P2, R3	0014:	0600ff03	00000044
L:	ADD R4, R1, R4	001C:	00040104	
	ADD R1, R2, R1	0020:	00010201	
	SUB R3, R1, R5	0024:	01030105	
	BNZ L	0028:	0802ff00	0000001C
	ST R4, P	0030:	0704ff00	00000048
	HLT	0038:	09000000	
P0:	.WORD 0	003C:	00000000	
P1:	.WORD 1	0040:	00000001	
P2:	.WORD 1	0044:	0000000a	
P:	.WORD 0	0048:	00000000	

Basic Instruction Cycle

Instruction Fetch-and-Execute cycle

- Instruction Fetch: Fetch instruction from Memory
- Instruction Execute: Execute the fetched instruction

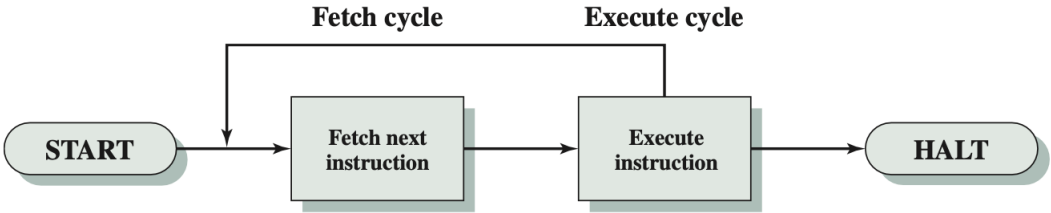


Figure 3.3 Basic Instruction Cycle

Instruction Execution

A more detailed look of Fig 3.3

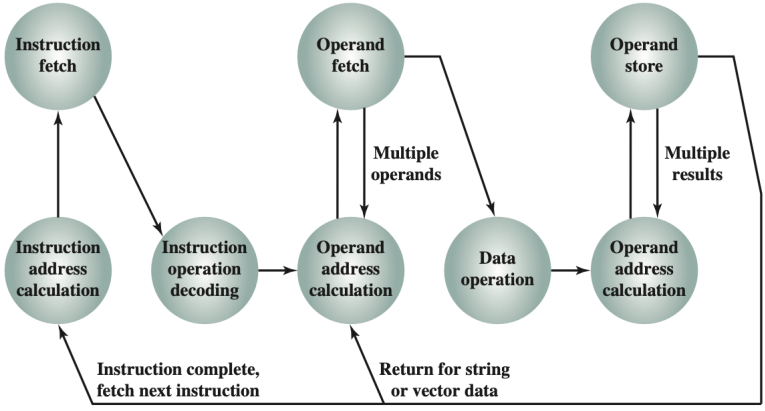


Figure 3.6 Instruction Cycle State Diagram

Instruction Address Calculation (iac)

- to determine the address of next instruction, usually this address is stored in a special register called **Program Counter (PC)**
- PC is automatically increased during the execution cycle.
- If the instruction is 4 bytes and the memory is byte addressable, Then add 4 to the previous address.
- For a two-word instruction, the second word is pointed by the PC, and the PC is increased again after reading the second word, so that it will point to the next instruction.

Instruction Address Calculation (iac)

- Most of the cases, implicit. explicit reference: next instruction reference in the instruction.
- PC is changed when you execute an instruction involving transfer of control, such as goto, if-then-else, function call, etc.

Instruction Fetch (if)

- Fetch the instruction from memory, with the address given by PC.
- Involves the following sequence of data movement:

$$\begin{aligned} \text{MAR} &\longleftarrow \text{PC (PC contains the address of next instruction)} \\ \text{IR} &\longleftarrow \text{mem[MAR] (Memory Read)} \end{aligned}$$

- MAR is the **Memory Address Register**, holding the address for memory operation.
- mem[MAR] is the **content** of the memory at location given by MAR, using memory read operation.
- IR is the **Instruction Register**, holding the instruction to be executed.

Instruction Decode

- Instructions are in the form of 32-bit binary pattern, e.g.,

```
0000 0110 0000 0000 1111 1111 0000 0100
```

- The leftmost 8 bits represents the operation to be performed (say `ADD`), and each subsequent 8 bits represents the location of operands (2 source operands and 1 destination operand).
- The control unit will decode the operations to be performed and set up the necessary operations (data movement/data processing).
- These operations will be dispatched by the control unit at the appropriate time during execution.

Operand Address Calculation

- The operands can either be in Memory or in Register (inside CPU)
- Registers are referenced by register number, e.g. R11 (register 11), no calculation needed.
- This is obtained from byte 1-3 of the instruction, and fed to the register file for reading the register content. `ADD R2, R3, R1`
- We need an address for memory operand.
- Sometimes the actual address is not provided, instead calculation may be needed. e.g. the address = the content of register 10 + A.
- We may need ALU operation (ADD, SUB, etc) to calculate the actual address of the operand.

Operand Fetch

- Fetch the operands to some temporary registers inside the CPU. (Data Movement)
- Most operation requires 2 source operands.
- The operand can be in memory or register (inside CPU).
- If an operand in memory, need a memory read operation, i.e. move the calculated address of the operand to MAR, perform a memory read.

$$\text{MBR} \longleftarrow \text{mem}[\text{MAR}]$$

  The **Memory Buffer Register (MBR)** holds the result of memory read. The data is then moved from MBR to the appropriate temporary register. (e.g. the input of the ALU)
- If operand is in register which is inside CPU, just read it from the register file with the register number as address, and then perform a data move.

Operand Fetch— Registers

- For example, for the instruction `ADD R4,R5,R6`. The address of first operand (i.e. 4) is fed to the register file (RF), and a RF read is performed.
- The data is then moved via system bus to the input1 of ALU.
- the address of second operand (i.e. 5) is fed to the RF, and a RF read performed. The data is moved via system bus to the input2 of ALU.

Operand Fetch— Memory

- It is a two-word instruction, `0x0600ff05 0000003c`, addresses are these two words are 300, and 304
- For the instruction `LD 0x0000003c, R4`.
- After instruction fetch, the program counter will point to the next word, i.e. 0000003c.
- Can read this address from memory by:

$$\text{MAR} \longleftarrow \text{PC}$$

MAR contains address of `0x0000003c`, MAR=304

$$\text{MBR} \longleftarrow \text{mem}[\text{MAR}]$$

reading memory, MBR= `0x0000003c`

Operand Fetch— Memory

- However `0x0000003c` is not operand, it is an address, we need to read memory again.
- To read the operand, we have to read the address `0000003c` by

$$\text{MAR} \longleftarrow \text{MBR}$$
$$\text{MBR} \longleftarrow \text{mem}[\text{MAR}]$$

Execute

- Perform Arithmetic/Logic operation for the instruction
- Source operands are already in some temporary storage inside CPU.
- These values are fed to the Arithmetic-Logic-Unit (ALU) for data processing.
- Output of the ALU (result of the operation) is then stored in some temporary register.

Operand Write Back

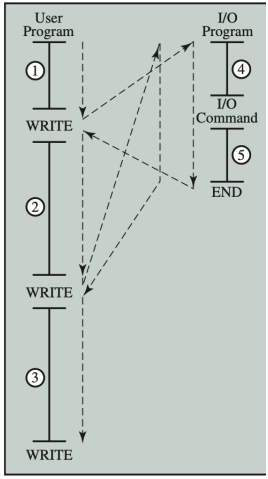
- If the destination operand is in register, then just move the data from ALU output (for ALU operations) to the input of RF.
- The RF address is in byte 3 of the instruction, which is fed to the RF.
- Perform a RF write. For memory operand, similar to operand fetch, calculation of address may be required.
- Move the data to MBR.
- Write back the destination operand to memory by:

$$\mathrm{mem}[\mathrm{MAR}] \longleftarrow \mathrm{MBR}$$

The address of memory is put in MAR, and the data to be written put in MBR.

Interrupt

In Fig 3.6, there is no way to escape from program execution.



(a) No interrupts

- Improve efficiency
- When an I/O arrives, it may need immediate attention of the CPU, e.g. data from network. Otherwise, there will be data loss.
- Other program may need CPU attention, e.g. in a time-sharing system, the CPU is shared among different programs.
- Need a way to stop the instruction execution process, and switch to another program — **Interrupt**.

Interrupt

- I/O devices will generate an interrupt signal to the CPU, telling the CPU that it wants attention.
- When processing an interrupt, the system need to remember where the original program was executing, then serve the interrupt, and return to the exact location to continue execution.
- you do not know when interrupt comes.
- Hence, the system need to be able to restore all system states, (e.g. The values of registers, etc.) so that the program can be continued as if nothing has happened.
- Interrupt handler can be hardware interrupts, software interrupts (operating system level)

Interrupt Handling

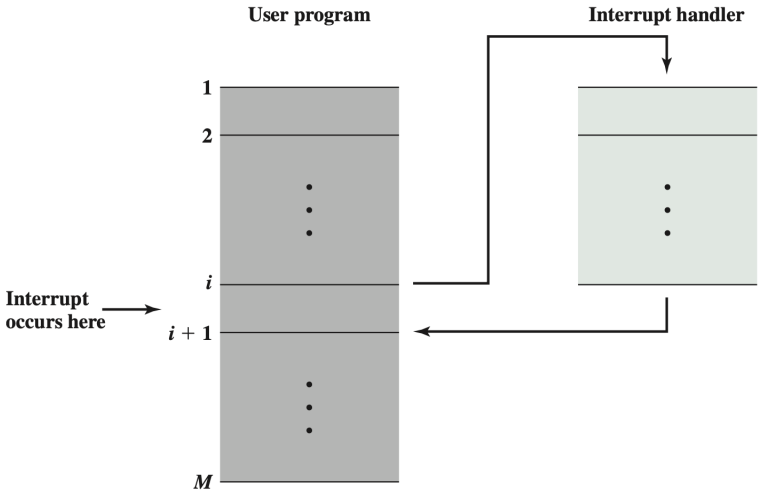


Figure 3.8 Transfer of Control via Interrupts

Interrupt Handling

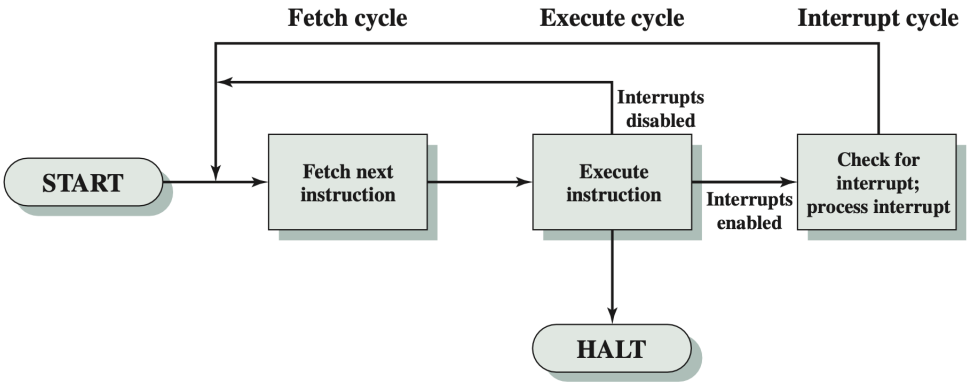


Figure 3.9 Instruction Cycle with Interrupts

Interrupt Handling

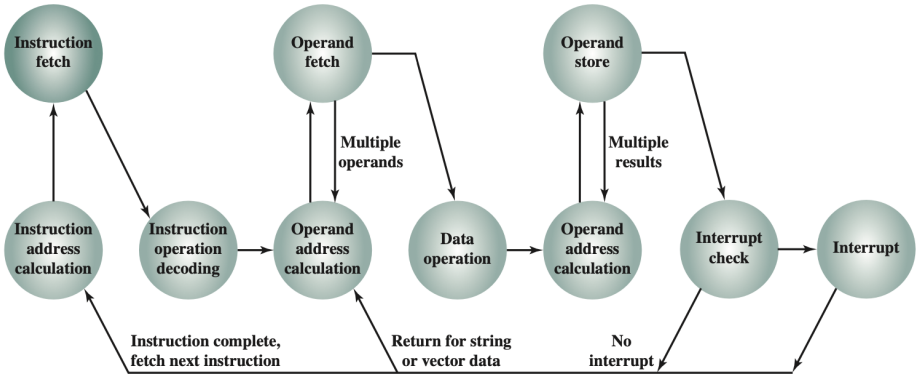


Figure 3.12 Instruction Cycle State Diagram, with Interrupts