EECE 321: Computer Organization

Mohammad M. Mansour
Dept. of Electrical and Compute Engineering
American University of Beirut

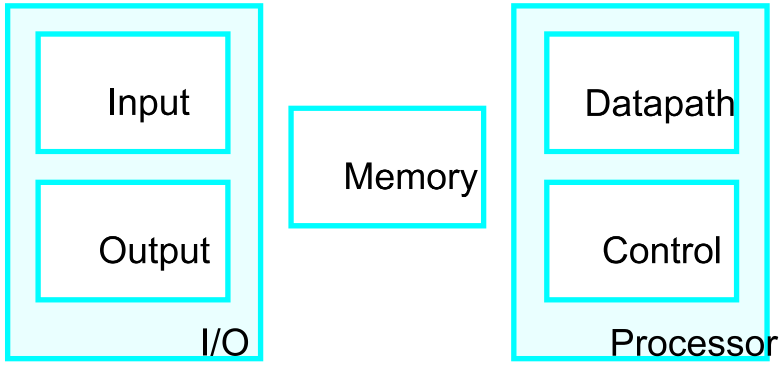
Lecture 3: Machine Instructions

Announcements

- HW1 due next Monday
  - Submit on Moodle
- Reading assignment
  - Ch.1 P&H
- Office hours:

Five Main Components of a Computer

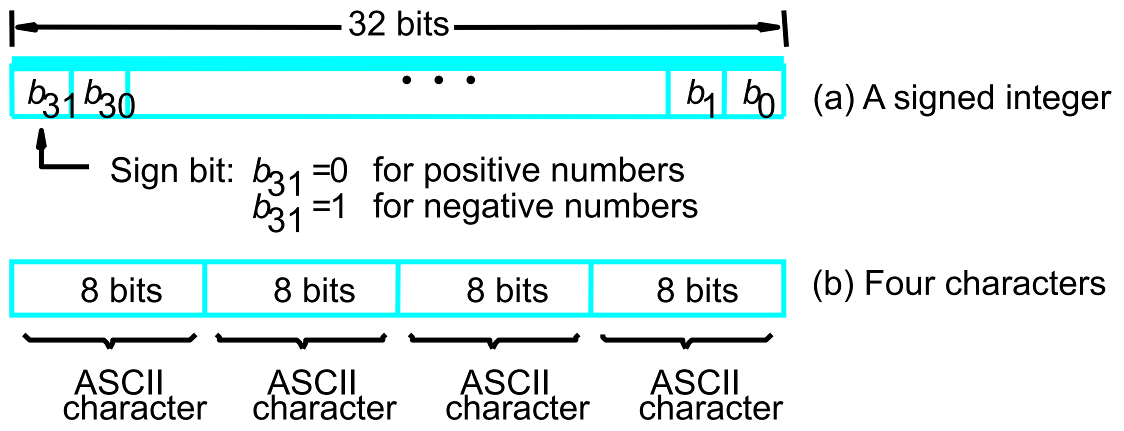
- Five functionally independent main components:
  - Input, memory, datapath, control, output

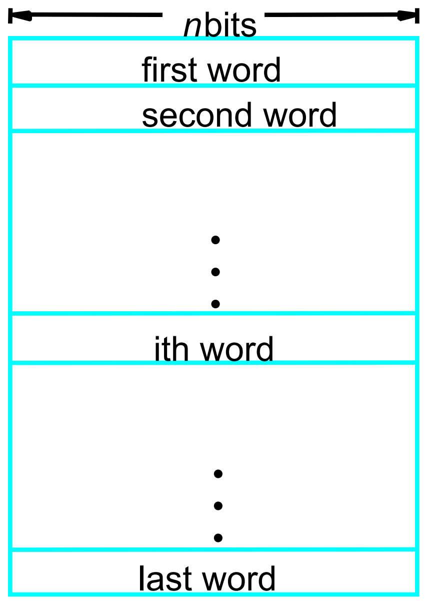


- Information handled by a computer is categorized as instructions or data.
- Instructions:
  - Instructions are explicit commands that
  - Govern transfer of info within a computer, between computer and its IO devices
  - Specify the arithmetic and logic operations to be performed
  - A list of instructions is called a program.
  - A program is stored in memory.
  - The processor fetches the instructions of a program from memory one after the other and performs the desired operations.
  - The computer is completely controlled by the stored program.
- Date: Are numbers used as operands to instructions.

Memory Locations and Addresses

- Memory stores groups of related bits (words) of size *n* called the word length.
- Modern computers have word lengths ranging from 16 to 64 bits.
- Ex: *n*=32, a word can be a 32-bit 2's complement arithmetic number
  - It can also be 4 ASCII characters (an ASCII character is represented in 8 bits)
  - A unit of 8 bits is called a byte

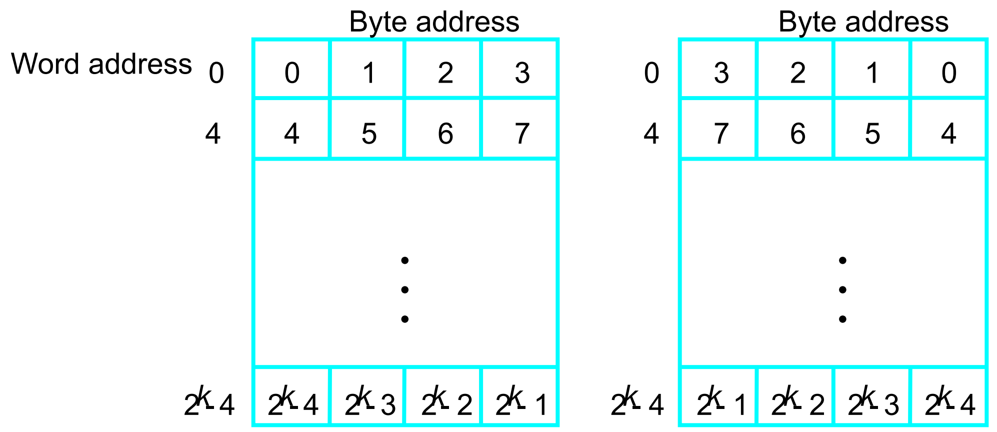




- Memory can be accessed in words or bytes.
- Word addressability: Send an address between 0 and 2^k-1
  - The 2^k addresses constitute the address space, where k = # of address bits.
  - Ex: A 32-bit address generates 2^{32} addresses (4G locations).

Byte Addressability

- Memory can be accessed in bytes with successive addresses referring to successive byte locations in memory (instead of words). [*byte addressable*]
- This assignment is typically used in modern computers. Ex: 32-bit address
  - Byte locations have byte addresses 0,1,2, …
  - Word locations have byte addresses 0,4,8, …
- There are two ways in which byte addresses can be assigned across words:
  - Big-Endean: lower byte address on the left (more significant)
  - Little-Endean: lower byte address on the right (less significant)
  - Both are used in commercial machines: Intel 80X86 little, Macintosh/Sun big



Big-Endean assignment   Little-Endean assignment

- Word alignment: Words are said to be aligned in memory if they begin at a byte address that is a multiple of the number of bytes in the word.
  - Ex: n=64, aligned words begin at byte addresses 0, 8, 16, …

Memory Operations: Load and Store

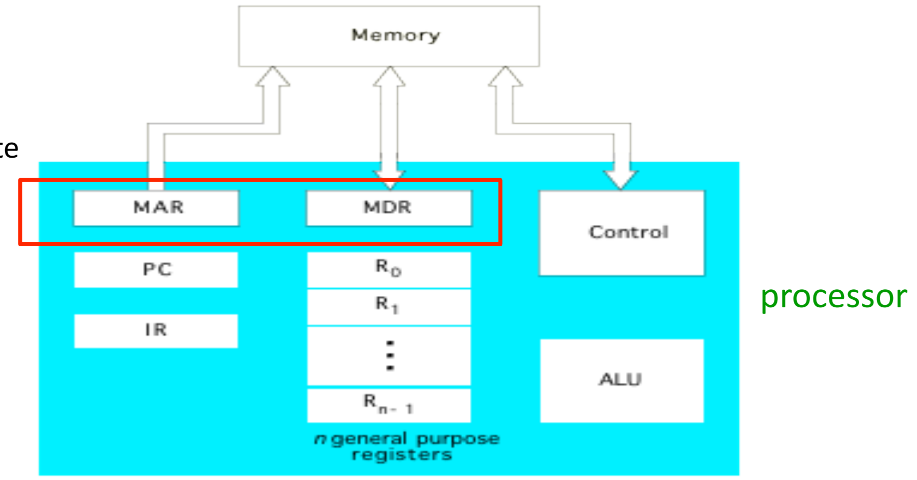
- The two basic memory operations that move data between processor and memory are Load and Store.
- Load (or fetch): Transfers a copy of the contents of a specific memory location to the processor.
  - The processor initiates a load operation by sending the required address and asserting the Read signal.
  - The memory sends back the contents of the addressed location to the processor where it is stored in a register.
- The processor uses a load operation to
  - Load the instructions of a program to be executed (instruction fetch)
    - Ex: Instructions are typically loaded from Instruction Cache
  - Load data on which these instructions operate, if any (operand fetch)
    - Ex: Data operands are typically loaded from Data Cache

Memory Operations: Store

- Store: Transfers information (data) from processor to a specific location in memory, destroying the old contents of that location.
  - The processor initiates a store operation by sending the address of the location together with the data to be stored, and asserting the Write Signal.
- The processor uses a store operation to write back the results (if any) of an instruction.
  - Store results back into Data Cache
- Remarks:
  - Memory operations take much more cycles to execute than ALU operations.
  - The processor (via the compiler) tries to minimize the number of Load/Store operations by storing frequently used operands in internal registers.

Example of Processor Internal Registers

MAR & MDR communicate with memory



- A processor uses registers internally to store instructions and data operands:
  - General purpose registers: R0, R1, … can be accessed by user.
    - These registers typically reside in a *Register File*
  - Special purpose register used exclusively by processor in controlling/executing instructions.
    - MDR: Memory Data Register
    - MAR: Memory Address Register
    - IR: Instruction Register
    - PC: Program Counter
- Data and instructions of a program are stored in memory externally.
  - Processor has to fetch them internally into its registers in order to operate on them.
- Remark: Don’t mix up registers with external memory.

Basic Operational Concepts

- A typical instruction looks like:
  - [instruction name] [one or more operands]
  - Each instruction has a name.
  - Operands can be register addresses and/or memory addresses.
- `Ex: Add R0,LOCA`
  - It adds the operand at memory location LOCA to the operand in a register R0 in the processor. The result is placed in R0.
  - This is called an ADD instruction. It combines a memory access operation and an arithmetic operation.
  - Time to execute instruction = time to access memory + time to add.
- Modern microprocessors implement these 2 operations using two separate instructions: Load followed by an Add.
  - `Load Rtemp, LOCA`
  - `Add  R0,Rtemp`
- A special register called Program Counter (PC) keeps track of the execution of a program. It contains the memory address of the next instruction to be fetched.
- The Instruction Register (IR) holds the instruction that is being executed.
- Communication between memory and processor is done using 2 registers:
  - Memory Address Register (MAR) holds the address of location to be accessed.
  - Memory Data Register (MDR) contains data to be written into or read out of the addressed location.

Typical Operating Steps

- Assume the instructions of a program are stored in memory (entered via Input).
- Execution starts when the PC is set to point to the 1st instruction of the program.
- The contents of the PC are sent to MAR and Read signal is sent to Memory.
- After Memory-Access time, the addressed word (1st inst. of the program) is loaded into MDR.
- The contents of MDR are transferred into IR. This completes the *instruction fetch* phase; the instruction is ready to be *decoded* and *executed*.
- The instruction is *decoded* to determine its type and its operands.
- For the previous Add instruction, it is necessary to obtain the first operand from memory (*operand fetch*). Its address is sent to MAR and the operand is fetched into MDR. The second operand is supplied by the register file.
- Operands are forwarded to the ALU which executes the instruction.
- The result needs to be *stored back* either in a register or in memory. For the latter case, the address of the destination is sent to MAR and the result is sent into MDR, and a Write control signal is asserted.
- At some point during execution of the current instruction, the *PC is incremented* to point to the next instruction to be executed.

Summary of Steps in Executing a Typical Instruction

- Instruction fetch
- Instruction decode
- Operand fetch
- Instruction execute
- Result write-back

Instructions and Instruction Sequencing

- The tasks carried out by a computer program consist of a sequence of small steps, such as adding two numbers, testing a condition, reading a character from the keyboard, or sending a character for display on a screen.

- A microprocessor must have instructions capable of performing 4 types of operations:
  - Data transfers between memory and the processor registers
  - ALU operations on data
  - Program sequencing and control (branches and jumps)
  - I/O transfers

- Assembly Language Notation to represent machine instructions and programs:

  INSTRUCTION_NAME   OPERANDS

  - Operands are either in registers (R1, R2, …) or in memory locations MEM[Address]
  - One of the operands should be the destination where the result is stored.
  - Convention: leftmost operand is the destination.

Instructions and Instruction Sequencing (cont'd)

- Example: Assume we want to execute the following statement in C language:
  - C = A + B // A,B,C are variables assigned to distinct memory locations.
- Let the variable name (A) designate the memory address, and the content of the addressed location represent the value of the variable (MEM[A]).
- The processor can't directly operate on variables while they are in memory.
- Use a pair of loads to fetch A and B into two registers:
  - `Load R1, A` // fetches the contents of A into register R1; R1 ← MEM[A]
  - `Load R2, B` // fetches the contents of B into register R2; R2 ← MEM[B]
- Now that the operands are available in registers, the processor can add them.
  - `Add R3,R1,R2   //R3←R1+R2,assuming a 3-address add instr.`
  - `Add R1,R2      //R1←R1+R2, assuming a 2-address add instr.`
- After the addition operation has been carried out, the sum is available in some register R.
- This sum must be stored back in memory at location C.
- Use a store instruction:
  - `Store R, C // stores R into memory address C; MEM[C]←R`

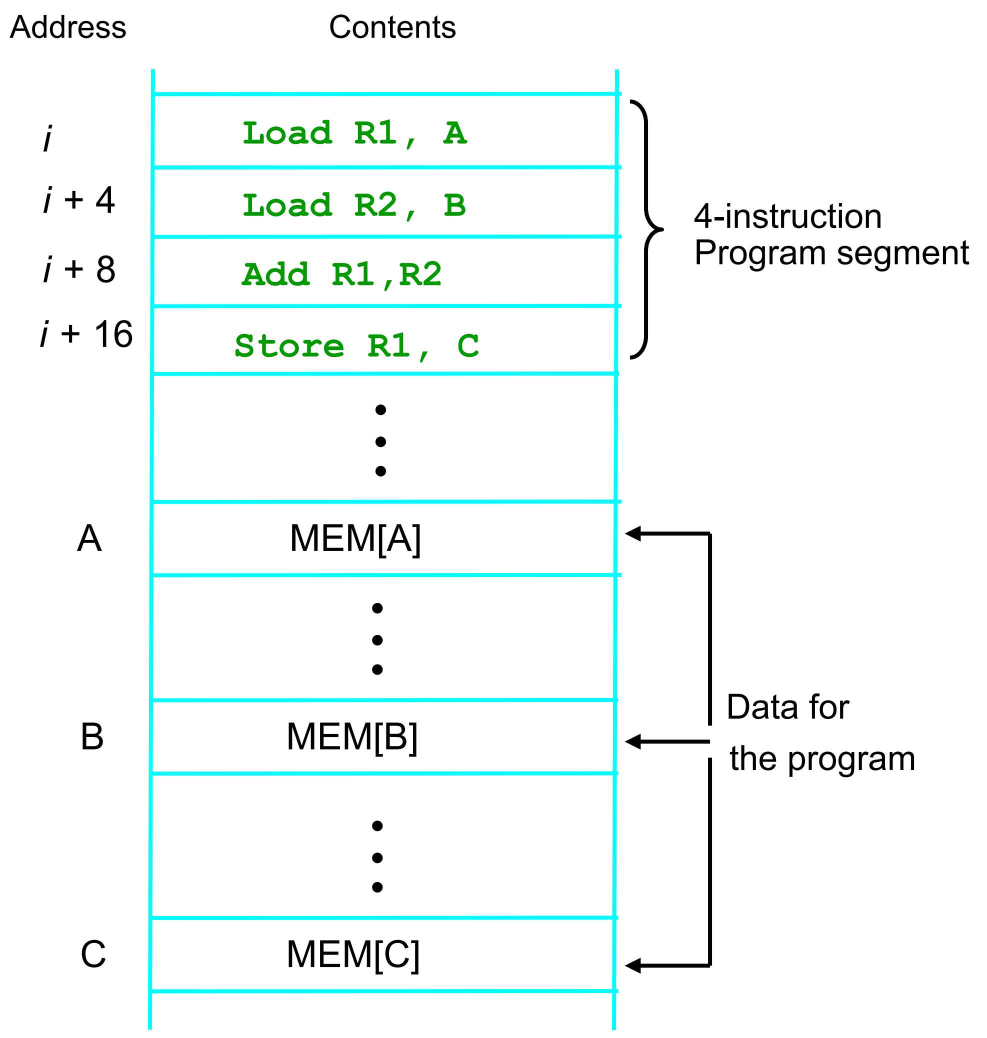
Instructions and Instruction Sequencing (cont'd)

C statement	Assembly language instructions
C = A + B	`Load R1, A`
	`Load R2, B`
	`Add R1, R2`
	`Store R1, C`

- How does the sequencing of instruction execution take place?
- Assume memory word length is 32 bits and memory is byte addressable.
- The 4 instructions are stored in successive word locations.
- The first instruction is stored in memory starting at address i.
- Since instructions are 4 bytes long, the other 3 instructions start at addresses i+4, i +8, i+16

Straight-Line Instruction Sequencing

- Register PC is initialized to address i.
- Instructions are fetched one after the other into IR for execution.
- This is referred to as straight-line sequencing.
- Note that the place where instructions are stored is separate from the place where data are stored.
- Five main phases of operations:
  - Instruction fetch
  - Instruction decode
  - Operand fetch
  - Instruction execute
  - Result write-back



Address
Contents
i
i + 4
i + 8
i + 16
Load R1, A
Load R2, B
Add R1,R2
Store R1, C
4-instruction Program segment
A
MEM[A]
B
MEM[B]
C
MEM[C]
Data for the program