Lecture 5: Computer Architecture

- Simpletron
  - Simpletron Architecture
  - Simpletron Instruction Set
  - Example Programs

The Simpletron is described in *C How to Program* *by Harvey Deitel.*

- Computer Architecture
  - Registers
  - Flags
  - Address Calculation

Simpletron Architecture

CPU

- The CPU contains one general-purpose register called the accumulator

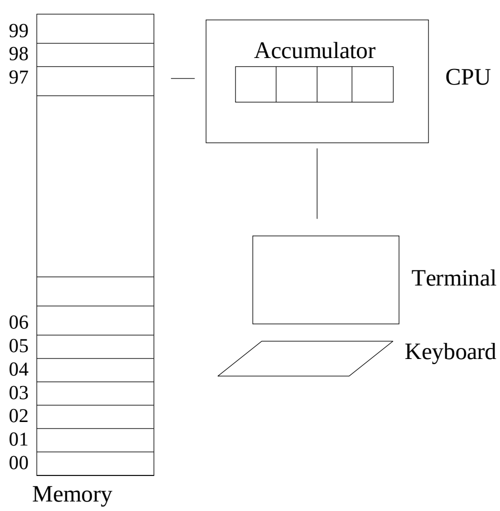
Memory

- All information in the Simpletron is handled in terms of words. A word is a signed four-digit decimal number such as +3364 or –0001
- The Simpletron is equipped with a 100-word memory, and these words are referenced by their location numbers 00-99
- Before running a Simpletron Machine Language (SML) program, it must be loaded into memory. The first instruction of every SML program is always placed in location 00.
- Each instruction occupies one word of memory. The sign of an SML instruction is always positive, but the sign of a data word may be either positive or negative. Each location in the Simpletron's memory may contain either an instruction, a data value used by the program, or an unused area of memory.

I/O

- The Simpletron uses a keyboard for input and a terminal screen for output

Simpletron



Instruction Set

Instruction	**Op code**	**Meaning**
READ	10	Reads a word from the terminal into a specific location in memory
WRITE	11	Writes a word from a specific location in memory to the terminal
LOAD	20	Loads a word from a specific location in memory into the accumulator
STORE	21	Stores a word from the accumulator into a specific location in memory
ADD	30	Adds a word from a specific location in memory to the word in the accumulator (leaves result in the accumulator)
SUBTRACT	31	Subtracts a word from a specific location in memory from the word in the accumulator (leaves result in the accumulator)

Instruction Set (cont.)

Instruction	Op code	Meaning
BRANCH	40	Branches to a specific location in memory
BRANCHNEG	41	Branches to a specific location in memory if the accumulator is negative
BRANCHZERO	42	Branches to a specific location in memory if the accumulator is zero
HALT	43	Halts

Simple Program

- Read two numbers, add them together, and print the sum.
- Algorithm:
  - read A
  - read B
  - sum = A + B
  - print sum
  - stop

Assembly Instructions

read A	(reads into the memory location used to store A)
read B	
load A	(puts A into the accumulator)
add B	(adds B to the accumulator, leaving the result in the accumulator)
store Sum	(saves the number in the accumulator into memory)
write Sum	(writes out the result to the terminal)
halt	

Simpletron Assembly Language → Simpletron Machine Language

- 1-1 Translation SAL->SML
- Execution starts at location 0
- In our example:
  - 7 instructions : locations 00 – 06
  - 3 data values
- Where to put data?
  - Directly after the program, or
  - In high memory, working down (99 and lower)

(homework hint: if you use the second option, then if you need to add instructions you can leave the data where it is and not have to re-do all your machine language that refers to it!)

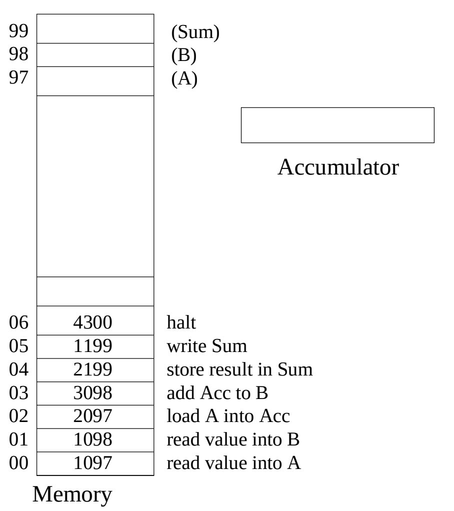
Translating into Simpletron Machine Code

- First, determine which memory locations you want to use for your data:
  - 97 – A
  - 98 – B
  - 99 – Sum
- Then, look up the opcodes for each instruction
  - 1097 read A (10 = read, 97 = location of A)
  - 1098 read B (10 = read, 98 = B location)
  - 2097 load A (20 = load, 97 = A location))
  - 3098 add B (30 = add, 98 = B location)

Translating, cont.

- 2199 store Sum (21 = store, 99 = sum location)
- 1199 write Sum (11 = write, 99 = sum location)
- 4300 halt (43 = halt)

Program Execution



Branching Example

Read two numbers from the keyboard and print the larger value:

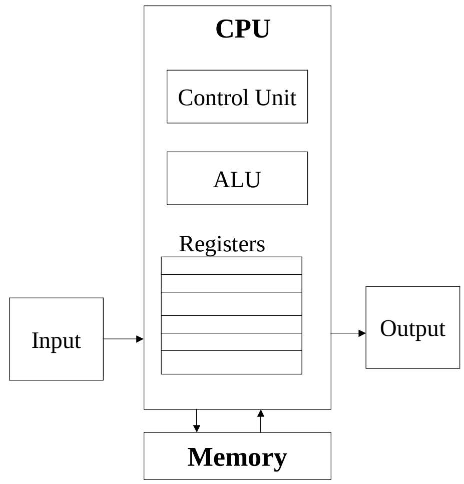
LOCATION	CONTENTS	MEANING
00	+1009	Read A
01	+1010	Read B
02	+2009	Load A
03	+3110	Subtract B
04	+4107	If B > A, go to 07
05	+1109	Write A
06	+4300	Halt
07	+1110	Write B
08	+4300	Halt
09	+0000	(Variable A)
10	+0000	(Variable B)

Looping Example

Use a loop to print the numbers one through 10:

LOCATION	CONTENTS	MEANING
00	+1107	Write the value of the variable Number
01	+2007	Load Number into the accumulator
02	+3008	Increment the accumulator by 1
03	+2107	Store incremented value back in Number
04	+3109	Subtract 11 from accumulator
05	+4100	Go to 00 if 10 iterations haven't been completed
06	+4300	All done; Halt
07	+0001	(Number)
08	+0001	Constant 1 (used for incrementing)
09	+0011	Constant 11 (loop limit)

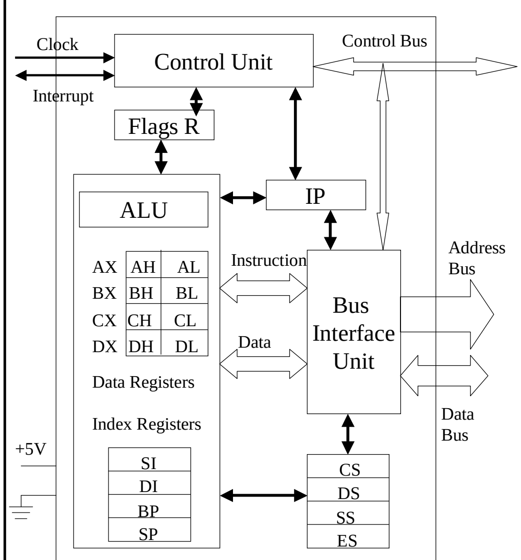
Computer Organization



Components

- Control Unit – fetches instructions, decodes instructions, causes instructions to be carried out.
- Arithmetic logical unit (ALU) – performs arithmetic operations (addition, etc.) on data.
- Registers – high speed memory cells (don't need to go through the bus to access). They vary in number and purpose on different machines.
- Buses – communication pathways connecting different devices/components.

8086 Block Diagram



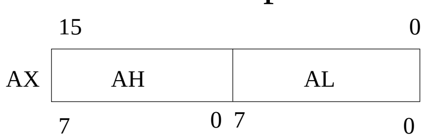
Registers

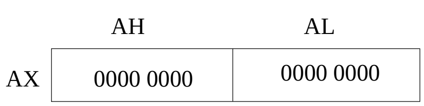
- 8, 16, or 32 bit high-speed storage locations inside the CPU
- They can be accessed at a much higher speed than conventional memory.
- When optimizing for speed, use registers.
- Four types: general purpose, segment, index, status, and control

General Purpose Registers

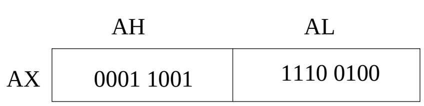
- Data registers, also known as general purpose registers: AX, BX, CX, DX
- Used for arithmetic operations and data movement
- Can be addressed as 16 bit or 8 bit values. For AX, upper 8 bits are AH, lower 8 bits are AL.
- Remember: when when a 16 bit register is modified, so is the corresponding 8 bit registers!

Example

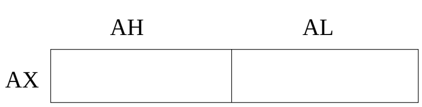




- move 0001 1001 1110 0100 to AX



- move 0011 1101 to AH



Special Attributes of GP Registers

- AX – accumulator – fastest for arithmetic operations. Some math instructions only use AX.
- BX – base – this register can hold an address of a procedure or variable. BX can also perform arithmetic and data movement.
- CX – counter – this register acts as a counter for repeating or looping instructions
- DX – data – this register has a special role in multiply and divide operations. In multiplication it holds the high 16 bits of the product. In division it holds the remainder.

Segment Registers

- Segment registers are used as base locations for program instructions, data, and the stack.
- All references to memory involve a segment register as the base location.

Segment Registers, cont.

- CS – code segment – this register holds the base location of all instructions in a program
- DS – data segment – this is the default base location for variables. It is used by the CPU to calculate the variable location.
- SS – stack segment – this register contains the base location of the stack.
- ES – extra segment – this is an additional base location for memory variables.

Index Registers

- Index registers contain the offsets of data and instructions.
- Offset refers to the distance of a variable, label, or instruction from its base segment.
- Index registers are used when processing strings, arrays, and other data structures.

Index Registers, cont.

- BP – base pointer – this register contains an offset from the SS register and is often used by subroutines to find the variables passed to it on the stack.
- SP – stack pointer – this register contains the offset from the top of the stack. The complete top of stack address is calculated using the SP and SS registers.
- SI – source index – used to point to data in memory. Named because this is the index register commonly used as the source in string operations (for example)
- DI - destination index – index register commonly used as the destination in string operations

Status and Control Registers

- IP – instruction pointer – always contains the offset of the next instruction. The IP and CS registers combine to form the complete address. IP is also known as PC – the program counter.
- Flags – a special register with individual bit positions that give the status of the CPU (control flags) or results of arithmetic operations (status flags).

Status Flags

- These indicate the status of arithmetic and logical operations.
- Carry flag (CF) – set if the result of an unsigned operation is too big to fit into the destination. 1 = carry, 0 = no carry.
- Overflow flag (OF) – set if the result of a signed operation is too wide to fit into the destination. 1 = overflow, 0 = no overflow.
- Sign flag (SF) – set when the result of an operation is negative. 1 = negative, 0 = positive

Status Flags, cont.

- Zero flag (ZF) – set when the result of an arithmetic operation is zero. Used by branch and loop instructions when comparing values. 1 = zero, 0 = not zero.
- Auxiliary Carry – set when an operation causes a carry from bit 3 to bit 4 or a borrow from bit 4 to bit 3. 1= carry, 0 = no carry.
- Parity – indicates if the result of an operation has an even or odd number of bits. Used to verify memory integrity or correct transmission of data.

Addressing

- Address: a number referring to an 8-bit memory location
- Logical addresses go from 0 to the highest location
- How these are translated into physical addresses varies.
- For Intel:
  - 32-bit segment-offset address: combination of base location (segment) and offset to represent a logical location
  - 20-bit absolute address, which refers to a physical memory location

Addressing, cont.

- Problem: how to address 1,048576 bytes of memory with a 16-bit wide address register (where the max is 65,535)
- Solution: combine segment and offset values to obtain the absolute address
- Example: 08F1:0100

1) convert segment to absolute by adding 4 zero bits: 08F10
2) add the offset: 0100 (hex)

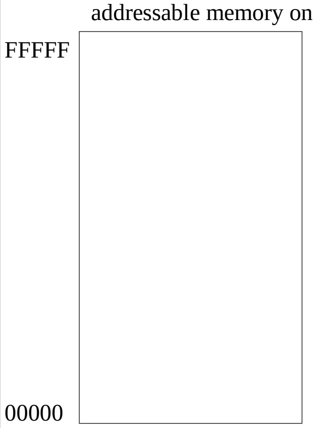
```
  0 8 F 1 0   -- segment value w/extra 4 0 bits
+ 0 1 0 0     -- add the offset
  0 9 0  1 0  -- obtain the absolute address
                 (effective address)
```

Why Segment-Offset?

- You can load the program at any segment address and individual variable addresses to not need to be recalculated.
  - Why? Variable locations are 16-bit offsets from the program's data area.
  - This is known as being *segment relocatable.*
- Programs can access large data structures by modifying the segment portion of the data's address to point to new blocks of memory.

Data Segment

addressable memory on 8086



Example:
DS = 0100h
data seg. start?
data seg. end?

If BX contains the offset:

BX = 005Ah
EA = ?

Segment Register Combinations

- Code Segment – the CS register and IP (instruction pointer) are used to point to the next instruction.
- Stack – the SS register is used with the SP (stack pointer) or BP (base pointer)
- Data Segment – DS with BX, SI, or DI
- Extended Segment – BX, SI, or DI

More on Effective Addresses

- There's more than one way to get the same effective address!
- Example:
  - CS = 147Bh
  - IP = 131Ah
  - EA = 147B0 + 131A = 15ACAh

  or
  - CS = 15ACh
  - IP = 000Ah
  - EA = 15AC + 000A = 15ACAh

- If CS = 147B, what range of effective addresses can be referenced without changing the value in CS?