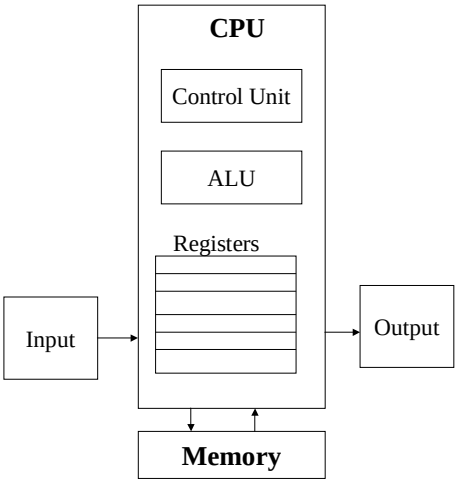
Lecture 23: Intro to Microprogramming

- Microprogramming Definition
- Machine Architecture Review
- Control Unit
- Instruction Cycle
- Microprogramming

Microprogramming

- Microprogramming is a method of implementing the behavior of machine instructions (produced by our assembler) by means of more elementary operations, in direct correspondence with the functions of the physical components of the computer.
- Microprogramming sees the computer at a greater level of detail than the computer architecture level.
- It needs to be able to specify state changes in the components of the physical structure.

Computer Organization



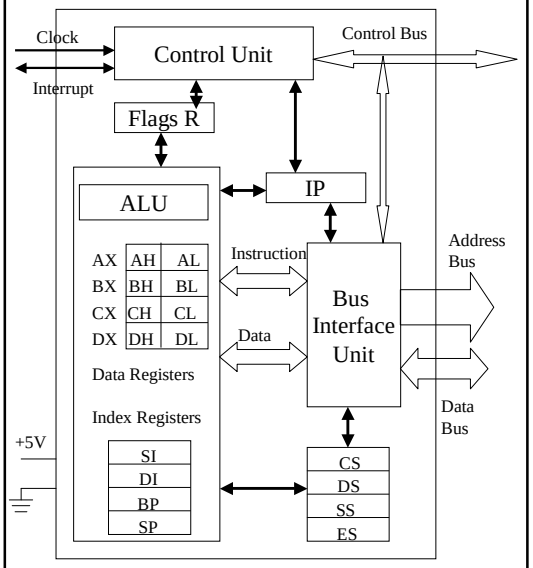
Components

- 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.

Memory Operation

- MAR (Memory Address Register) – drives the address bus
- MBR (Memory Buffer Register) – drives the data bus.
- Figure 4-6, Tannenbaum 3rd Edition

8086 Block Diagram



- What we've seen:
  - ALU (combinational circuit)
  - Registers (sequential circuits)
  - Memory (sequential circuits)
- What we haven't seen:
  - Control Unit

Control Unit

- We need to get from the machine code down to the control signals that regulate the gates and circuits.
- Early computers hardwired this.
- RISC machines do as well.
- Microprogramming is an alternative that allows for simpler machine hardware.

Hardwired vs. Microprogrammed Control

- Hardwired:
  - composed of combinatorial and sequential circuits that generate complete timing that corresponds with execution of each instruction.
  - time-consuming and expensive to design
  - difficult to modify
  - … but fast

Hardwired vs. Microprogrammed Control (cont.)

- Microprogrammed:
  - design is simpler – problem of timing each instruction is broken down. Microinstruction cycle handles timing in a simple and systematic way.
  - easier to modify
  - slower than hardwired control

Instruction Cycle

- Operation of a computer consists of a sequence of instruction cycles, one machine code instruction per cycle.
- Instruction cycles can be subdivided into:
  - fetch* - retrieving the instruction
  - indirect – retrieving indirect operands (if any)
  - execute* - executing the instruction
  - interrupt – handling any interrupts that may have occurred

    *always occur

we'll look mostly at the fetch-execute portion

Micro-Operations

- Each step of the instruction cycle can be broken down further into MicroOperations (μOps or μOperations)

Fetch-Execute Cycle

1. extract the instruction from memory
2. calculate the address of the next instruction, by advancing the PC (program counter)
3. decode the opcode
4. calculate the address of the operand (if any)
5. extract the operand from memory
6. execute
7. calculate the address of the result
8. store the result in memory

Fetch (steps 1-3)

- Brings the instructions in from memory to the IR (Instruction Register). Involves four registers:
  - MAR specifies address of read/write operation
  - MBR contains value to be read/written
  - PC contains address of next instruction to fetch

Fetch (steps 1-3)

- Sequence of events (at start, address of next instruction is in the PC)
  1. "latch" the address onto the MAR. The address can remain set on the address bus for the entire time required for the read operation, while it is possible to change the contents of the PC.
  2. bring in the instruction
     - address in MAR placed on address bus
     - control unit issues RD signal on control bus
     - result appears on data bus, is copied into MBR
  3. increment PC to get ready for the next instruction
  4. move contents of MBR to IR (frees up MBR for execute portion of the cycle)

Execute (Steps 4-8)

- Unlike Fetch, which is the same for all machine instructions, for a machine with N different opcodes there will be N different sequences of μOps that can occur!
- Also, not all of the five steps apply in each case:
  4. calculate the address of the operand (if any)
  5. extract the operand from memory (if any)
  6. execute
  7. calculate the address of the result (if storing)
  8. store the result in memory (if storing)

Execute Example

- ADD instruction:
  1. MAR = address portion of IR
  2. MBR = contents of memory at that address
  3. ACC (Accumulator) = ACC + MBR

Putting it All Together

- Our Micro-Ops need to be put together in an order that makes sense!
- The microprogram implements an algorithm responsible for sequencing the micro-ops (ensuring that they are executed in the proper order) and executing the micro-ops (carrying out the state changes involved in the instruction)

The Microprogram

- We need to be able to express the sequence of steps required to carry out our instruction cycle.
- Here's a set for fetch:

```
0:  MAR = PC;  RD;
1:  PC = PC +1; RD;
2:  IR = MBR;
```

- The semicolon indicates μOps that are issued at the same time.
- The carriage return (different lines) indicate operations that occur in sequence.
- Why two reads? (RD) – read (and write) takes two instruction cycles.