# Hardwired Control Unit
# Ch 16

Micro-operations

Controlling Execution

Hardwired Control

## What is Control (2)

- So far, we have shown what happens inside CPU
  - execution of instructions
    - opcodes, addressing modes, registers
    - I/O & memory interface, interrupts
- Now, we show how CPU controls these things that happen
  - how to control what gate or circuit should do at any given time
    - control wires transmit control signals
    - control unit decides values for those signals

## Micro-operations (2)

(mikro-operaatio)

- Basic operations on which more complex instructions are built
  Fig. 16.1 (Fig. 14.1 [Stal99])
  - each execution phase (e.g., fetch) consists of one or more sequential micro-ops
  - each micro-op executed in one clock cycle in some subsection of the processor circuitry
  - each micro-op specifies what happens in some area of cpu circuitry
  - system cycle time determined by longest micro-op!
- Many micro-ops (for successive instructions) can be executed simultaneously
  - if non-conflicting, independent areas of circuitry

## Instruction Fetch Cycle (10)

- 4 registers involved
  - MAR, MBR, PC, IR
- What happens?

Fig. 12.6 (Fig. 11.7 [Stal99])

| What happens | micro-ops? |
|---|---|
| Address of next instruction is in PC | MAR ← (PC) |
| Address (MAR) is placed on address bus | READ |
| READ command given to memory | |
| Result (from memory) appears on data bus | MBR ← (mem) |
| Data from data bus copied into MBR | PC ← (PC) +1 |
| PC incremented by 1 | IR ← (MBR) |
| New instruction moved from MBR to IR | |
| MBR available for new work | |

## Instruction Fetch Micro-ops (2)

- 4 micro-ops
  - can not change order, can do some ops at the same time
  - s2 must be done after s1
  - s3 can be done simultaneously with s2
  - s4 can be done with s3, but must be done after s2

s1: MAR ← (PC), READ
s2: MBR ← (mem)
s3: PC ← (PC) +1
s4: IR ← (MBR)

⇒ Need 3 ticks:

| | |
|---|---|
| t1: | MAR ← (PC) |
| t2: | MBR ← (mem) |
| | PC ← (PC) +1 |
| t3: | IR ← (MBR) |

implicit READ

assume: mem read in one cycle

## Micro-op Grouping (4)

- Must maintain proper sequence (semantics)
- No conflicts
  - no write to/read from with same register (set?) at the same time
  - each circuitry can be used by only one micro-op at a time
    - E.g., ALU or some bus

| | |
|---|---|
| t1: | MAR ← (PC) |
| t2: | MBR ← (mem) |

| | |
|---|---|
| t2: | MBR ← (mem) |
| t3: | IR ← (MBR) |

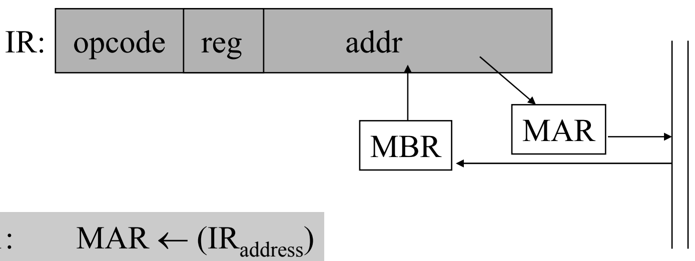
| | |
|---|---|
| t2: | PC ← (PC) + 1 |
| t3: | R1 ← (R1) + (MBR) |

## Micro-op Types (4)

- Transfer data from one reg to another
- Transfer data from reg to external area
  - memory
  - I/O
- Transfer data from external to register
- ALU or logical operation between registers

## Indirect Cycle

- Instruction contains address of an operand, instead of direct operand address



t1: MAR ← $(IR_{address})$
t2: MBR ← (mem)
t3: $IR_{address}$ ← (MBR)

(Replace indirect address by direct address)

## Interrupt Cycle

- After execution cycle, test for interrupts
- If interrupt bits on, then
  - save PC to memory
  - jump to interrupt handler
  - or, find out first correct handler for this type of interrupt and then jump to that (need more micro-ops)
  - context saved by interrupt handler

```
t1:    MBR ← (PC)
t2:    MAR ← save-address
       PC ← routine-address
t3:    mem ← (MBR)
```

implicit - just wait?

## Execute Cycle (4)

- Different for each op-code

```
t1:  ALU1 ← (R2)
     ALU2 ← (R3)
t2:  ALUout ← "+"
t3:  R1 ← ALUout
```

ADD R1, X

```
t1:    MAR ← (IR_address)
t2:    MBR ← (memory)
t3:    R1 ← (R1) + (MBR)
```

?

ADD R1, R2, R3

```
t1:    R1 ← (R2) + (R3)
```

JMP LOOP

```
t1:    PC ← (IR_address)
```

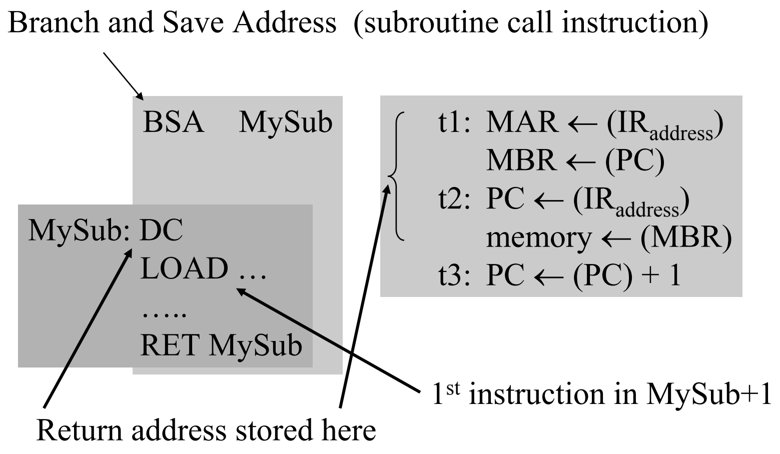
Was this updated in indirect cycle?

BZER R1, LOOP

```
t1:    if ((R1)=0) then
              PC ← (IR_address)
```

Can this be done in one cycle?

## Execute Cycle (contd) (1)


Branch and Save Address (subroutine call instruction)

```
BSA    MySub

MySub: DC
       LOAD …
       …..
       RET MySub
```

$t1$: MAR ← ($IR_{address}$)
MBR ← (PC)
$t2$: PC ← ($IR_{address}$)
memory ← (MBR)
$t3$: PC ← (PC) + 1

## Instruction Cycle (3)

- Decomposed to micro-ops
- State machine for processor
  - state: execution phase
  - sub-state: current group of micro-ops executable in one clock cycle (tick)
- In each sub-state the control signals have specific values dependent
  - on that sub-state
  - on IR register fields and on flags
    - including control signals from the bus
    - including values (flags) produced by previous sub-state

(Fig. 14.3 [Stal99])
Fig. 16.3

(Fig. 14.4 [Stal99])
Fig. 16.4

## Control State Machine (2)

- Each state defines current control signal values
  - determines what happens in next clock cycle

Control execution

- Current state and current register/flag values determine next state

Control sequencing

## Control Signal Types

- Control data flow from one register to another
- Control signals to ALU
  - ALU does also all logical ops
- Control signals to memory or I/O devices
  - via control bus

## Control Signal Example (5)

(Fig. 14.5 [Stal99])
Fig. 16.5

- Accumulator architecture
- Control signals for given micro-ops <u>cause</u> micro-ops to be executed

Table 16.1
(Tbl 14.1 [Stal99])

  – setting $C_2$ makes value stored in PC to be copied to MAR in next clock cycle
    - $C_2$ controls Input Data Strobe for MAR (see Fig. A.30 for register circuit)
  – setting $C_R$ & $C_5$ makes memory perform a READ and value in data bus copied to MBR in next clock cycle
  – micro-op = collection of control signals?

## Example: Intel 8085 (5)

- Introduced 1976
- 3, 5, or 6 MHz, no cache
- 8 bit data bus, 16 bit address bus
  – multiplexed
- One 8-bit accumulator

Fig. 16.7
(Fig. 14.7 [Stal99])

| | opcode | address | |
|---|---|---|---|
| LDA MyNumber | 0x3A | 0x10A5 | 3 bytes |
| OUT #2 | 0x2B | 0x02 | 2 bytes |
| | opcode | port | |

## Example: i8085 (6)

- Instead of complex data path all data transfers within CPU go via internal bus (Fig. 14.7 [Stal99]) Fig. 16.7
  - may not be good approach for superscalar pipelined processor - bus should not be bottleneck
- External signals Table 16.2 (Tbl 14.2 [Stal99])
- Each instruction is 1-5 machine cycles
  - one external bus access per machine cycle
- Each machine cycle is 3-5 states
- Each state is one clock cycle (Fig. 14.9 [Stal99])
- Example: OUT instruction Fig. 16.9

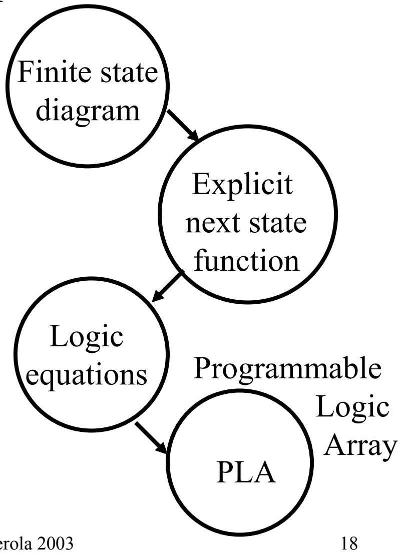
## Hardwired Control Logic Implementation (3)

Initial representation:

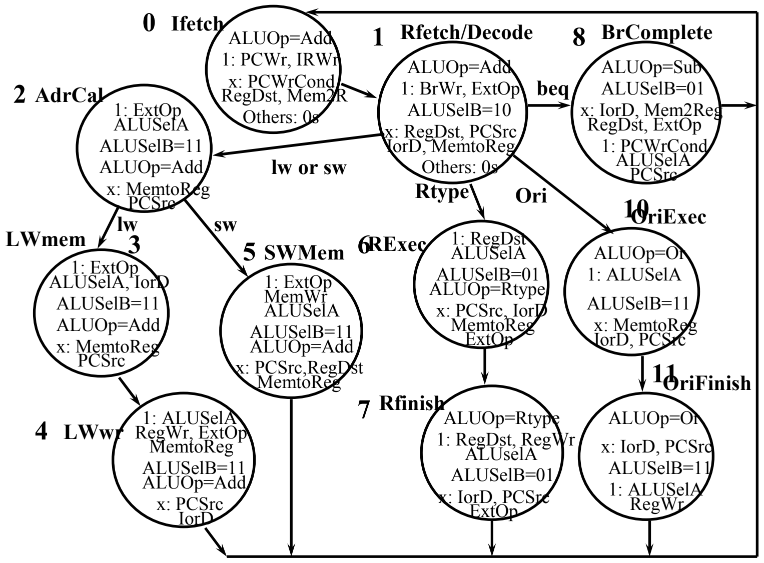
Sequencing control:

Logic representation:

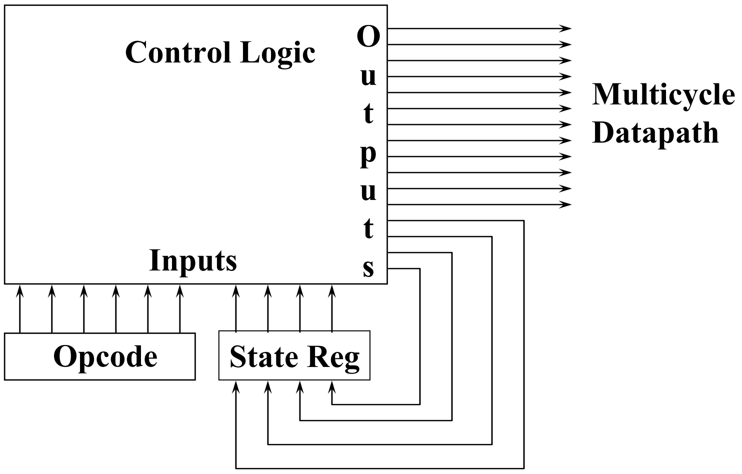
Implementation:



## Finite State Diagram



## Explicit Next State Function



## Logic Equations (2)

Next state from current state

- State 0 -> State1
- State 1 -> S2, S6, S8, S10
- State 2 -> S5 or …
- State 3 -> S9 or …
- State 4 ->State 0
- State 5 -> State 0
- State 6 -> State 7
- State 7 -> State 0
- State 8 -> State 0
- State 9-> State 0
- State 10 -> State 11
- State 11 -> State 0

Alternatively,
prior state & condition

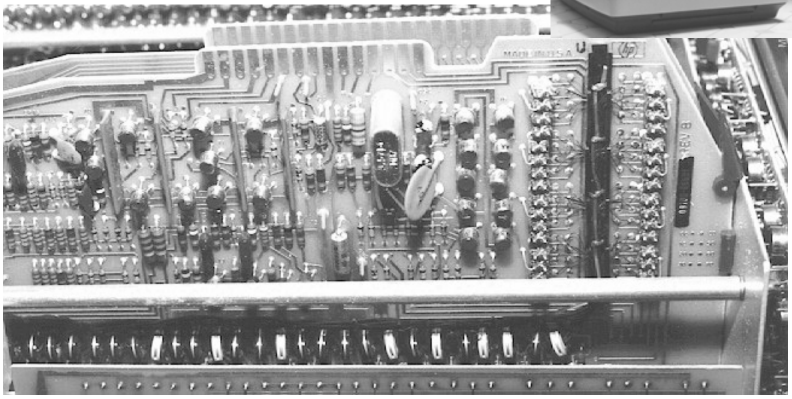
| | |
|---|---|
| S4, S5, S7, S8, S9, S11 | -> State0 |
| ________________ | -> State1 |
| ________________ | -> State 2 |
| ________________ | -> State 3 |
| ________________ | -> State 4 |
| State2 & op = SW | -> State 5 |
| ________________ | -> State 6 |
| State 6 | -> State 7 |
| ________________ | -> State 8 |
| State3 & op = JMP | -> State 9 |
| ________________ | -> State 10 |
| State 10 | -> State 11 |

## Hardwired Control Logic (3)

- Circuitry becomes very big and complex very soon
  - may be unnecessarily slow
  - simpler is smaller, and thus faster
- Many lines (states) exactly or almost similar
- Have methods to find similar lines (states) and combine them
  - not simple
  - save space, may lose in speed
  - must be redone after any modification

# -- End of Chapter 16: Hardwired Control --

HP 9100 Calculator (1968), 20 kg,
$5000, 16 regs (data or 14 instructions/reg),
32Kb ROM, 2208 bit RAM magnetic core memory



Hardwired Control Logic board http://www.hpmuseum.org/9100cl.jpg