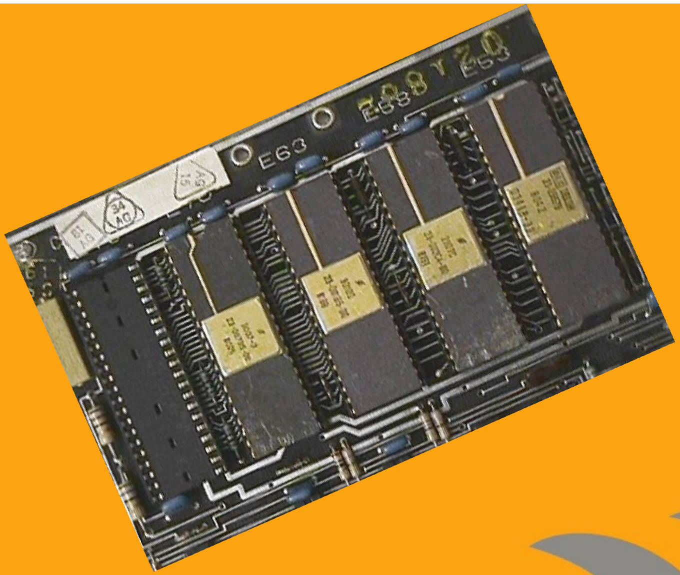
Lecture 10

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI



Control Unit (*Ohjausyksikkö*)

Ch 15-16 [Sta10]

- Micro-operations
- Control signals (*Ohjaussignaalit*)
- Hardwired control (*Langoitettu ohjaus*)
- Microprogrammed control (*Mikro-ohjelmoitu ohjaus*)

What is Control?

- Architecture determines the CPU functionality that is visible to 'programs'
  - What is the instruction set ?
  - What do instructions do?
  - What operations, opcodes?
  - Where are the operands?
  - How to handle interrupts?

Functional requirements for CPU

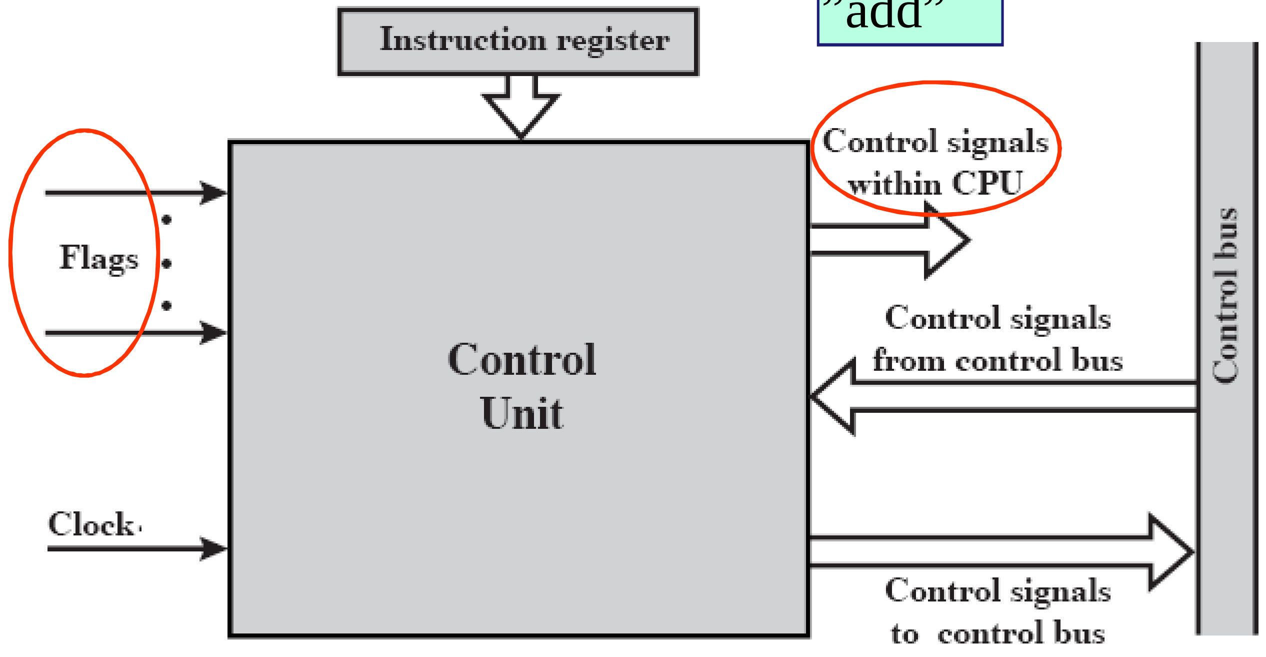
1. Operations
2. Addressing modes
3. Registers
4. I/O module interface
5. Memory module interface
6. Interrupt processing structure

- Control Unit, CU (*ohjausyksikkö*) determines how these things happen in hardware (CPU, MEM, bus, I/O)
  - What gate and circuit should do what at any given time
  - Selects and gives the control signals to circuits in order
  - Physical control wires transmit the control signals
    - Timed by clock pulses
    - Control unit decides values of the signals

Control Signals

”read”
”write”
”add”

(Sta10 Fig 15.4)


Instruction register
Flags
Clock
Control Unit
Control signals within CPU
Control signals from control bus
Control signals to control bus
Control bus

- Main task: control data transfers
  - Inside CPU: REG ⇔ REG, ALU ⇔ REG, ALU-ops
  - CPU ⇔ MEM (I/O-controller): address, data, control
- Timing (*ajoitus*), Ordering (*järjestys*)

Micro-Operations

- Simple control signals that cause one very small operation (*toiminto*)
  - E.g. Bits move from reg 1 through internal bus to ALU
- Subcycle duration determined from the longest operation
- During each subcycle multiple micro-operations in action
  - Some can be done simultaneously,
    - If in different parts of the circuits
  - Must avoid resource conflicts
    - WAR or RAW, ALU, bus
  - Some must be executed sequentially

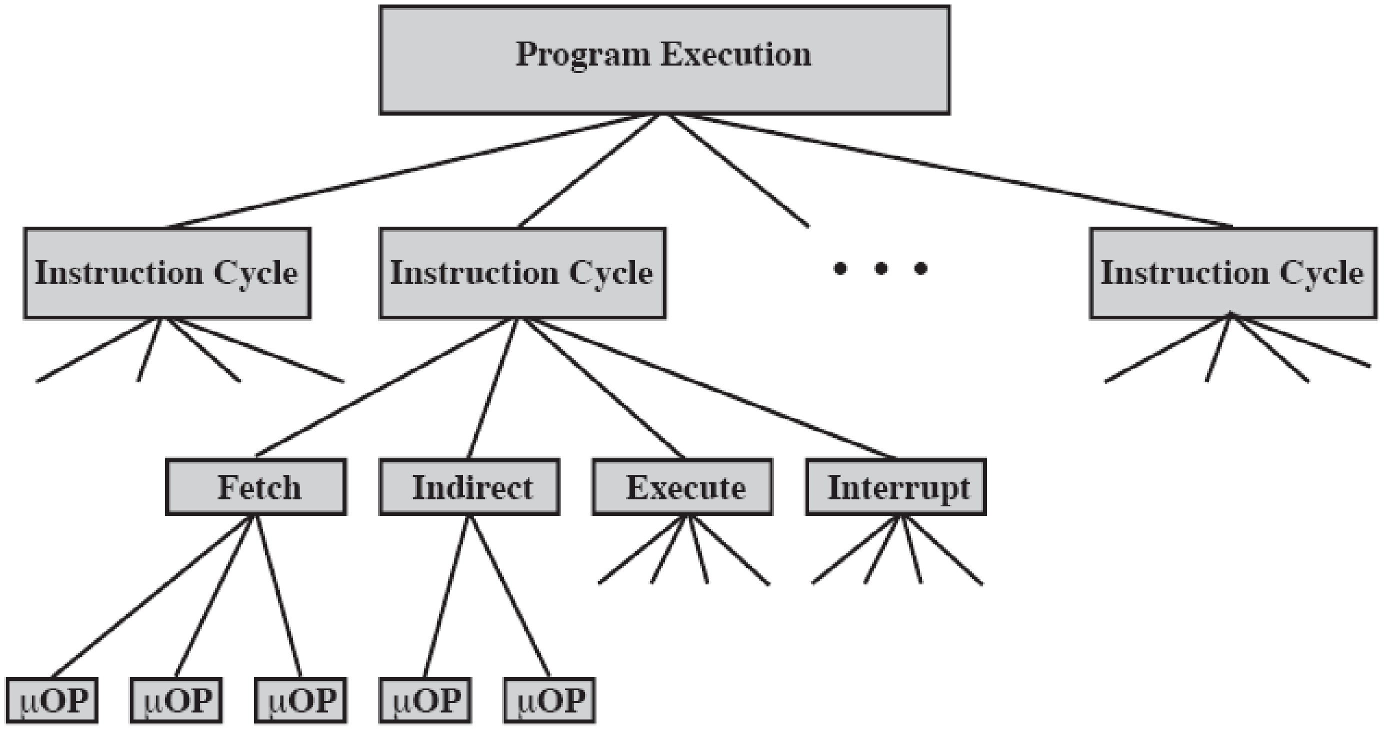
    to maintain the semantics

t1: MAR $\leftarrow$ PC
t2: MBR $\leftarrow$ MEM[MAR]
$\boxed{\text{PC} \leftarrow \text{PC} + 1}$
t3: IR $\leftarrow$ (MBR)

If implemented without ALU

Instruction cycle (Käskysykli)

(Sta10 Fig 15.1)



- When micro-operations address different parts of the hardware, hardware can execute them parallel
- See Chapter 12 instruction cycle examples (next slide)

Instruction Fetch Cycle

Example:

t1: MAR ← PC

t2: MAR ← MMU(MAR)

Control Bus ← Reserve

wait?

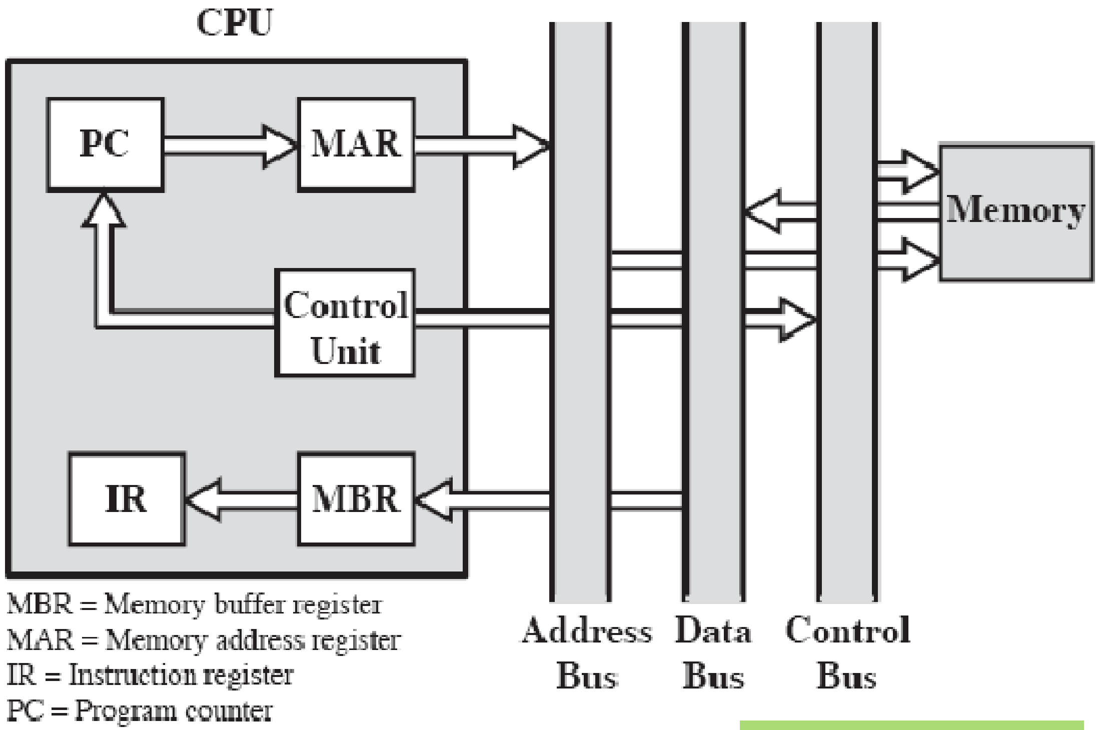
t3: Control Bus ← Read

PC ← PC + 1

t4: MBR ← MEM[MAR]

Control Bus ← Release

t5: IR ← MBR



MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

(Sta10 Fig 12.6)

Execution order? What can be executed parallel?
Which micro-ops to same subcycle,
which need own cycle?

Instruction Cycle

- Operand fetch cycle(s)
  - From register or from memory
  - Address translation
- Execute cycle(s)
  - Execution often in ALU
  - Operands in
    and control operation
  - Result from output
    to register /memory
  - flags ← status
- Interrupt cycle(s)
  - See examples (Ch 12): Pentium
  - What to do using same micro-operation?
  - What micro-ops parallel / sequentially?

<u>ADD r1, r2, r3:</u>

```
t1: ALUin1 ← r2
t2: ALUin2 ← r3
      ALUoper ← IR.oper
t3: r1 ← ALUout
      flags ← xxx
```

<u>ISZ X, Increment and Skip if zero:</u>

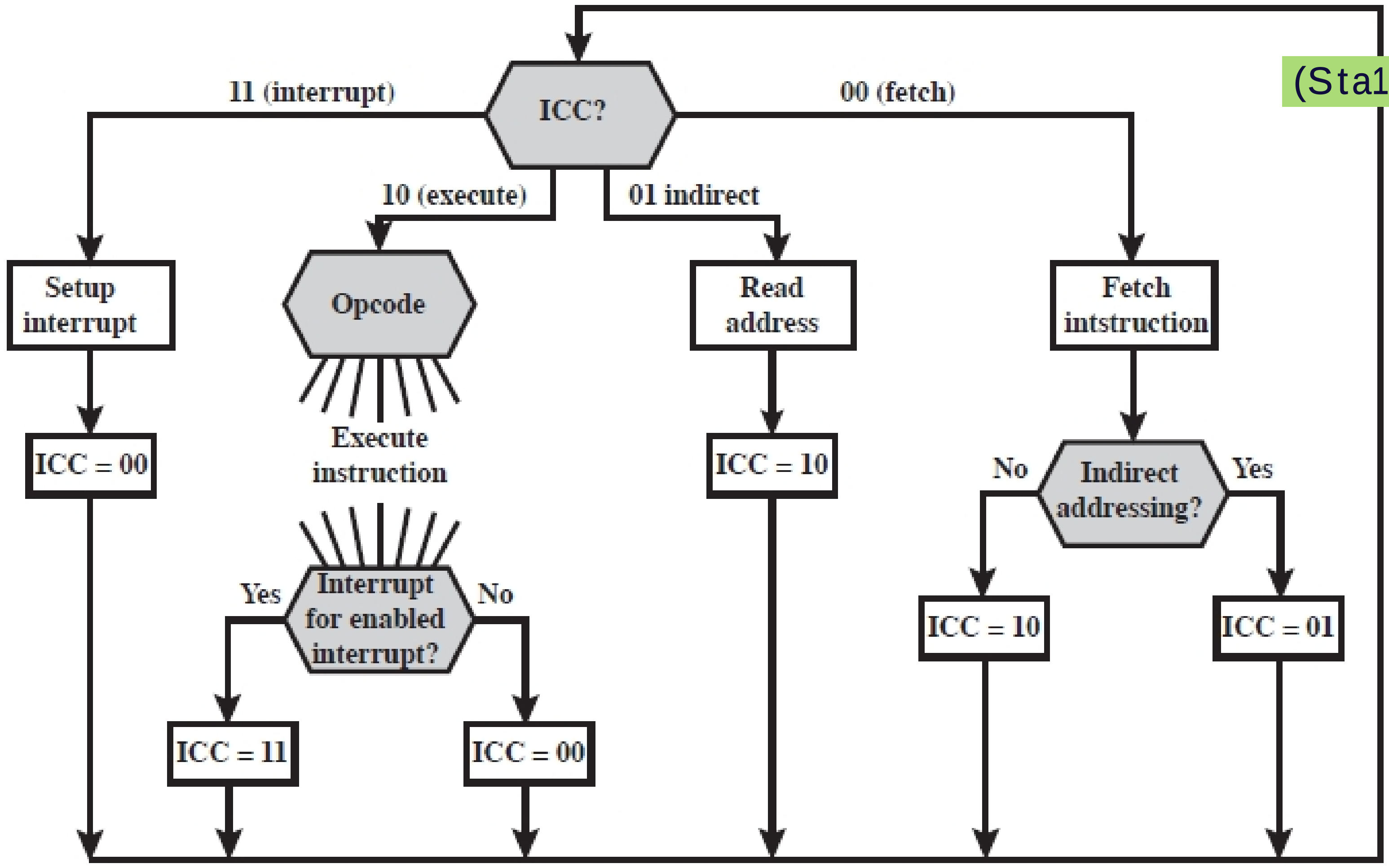
```
t1: MAR ← IR.address
t2: MBR ← MEM[MAR]
t3: MBR ← MBR+1
t4: MEM[MAR] ← MBR
      if (MBR=0) then PC ← PC +1
```

Conditional operation possible

Instruction Cycle Flow Chart as State-Machine

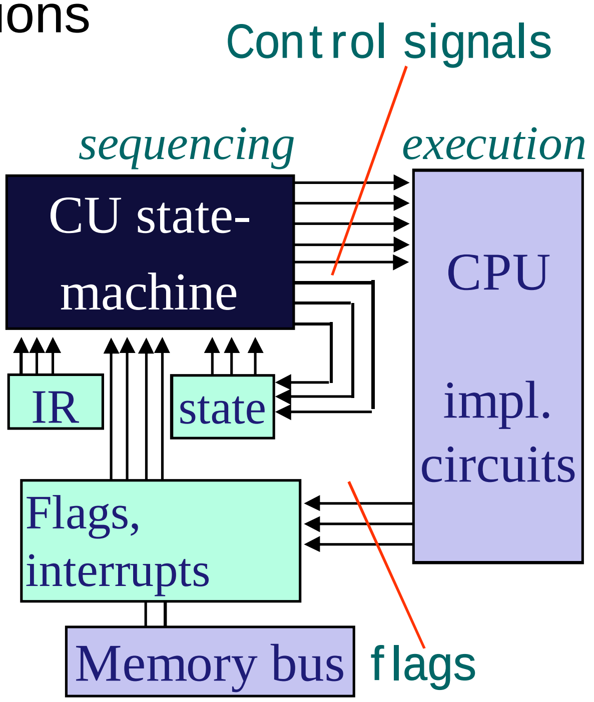
- ICC: Instruction Cycle Code register's state

(Sta10 Fig 15.3)



Instruction Cycle Control as State-Machine

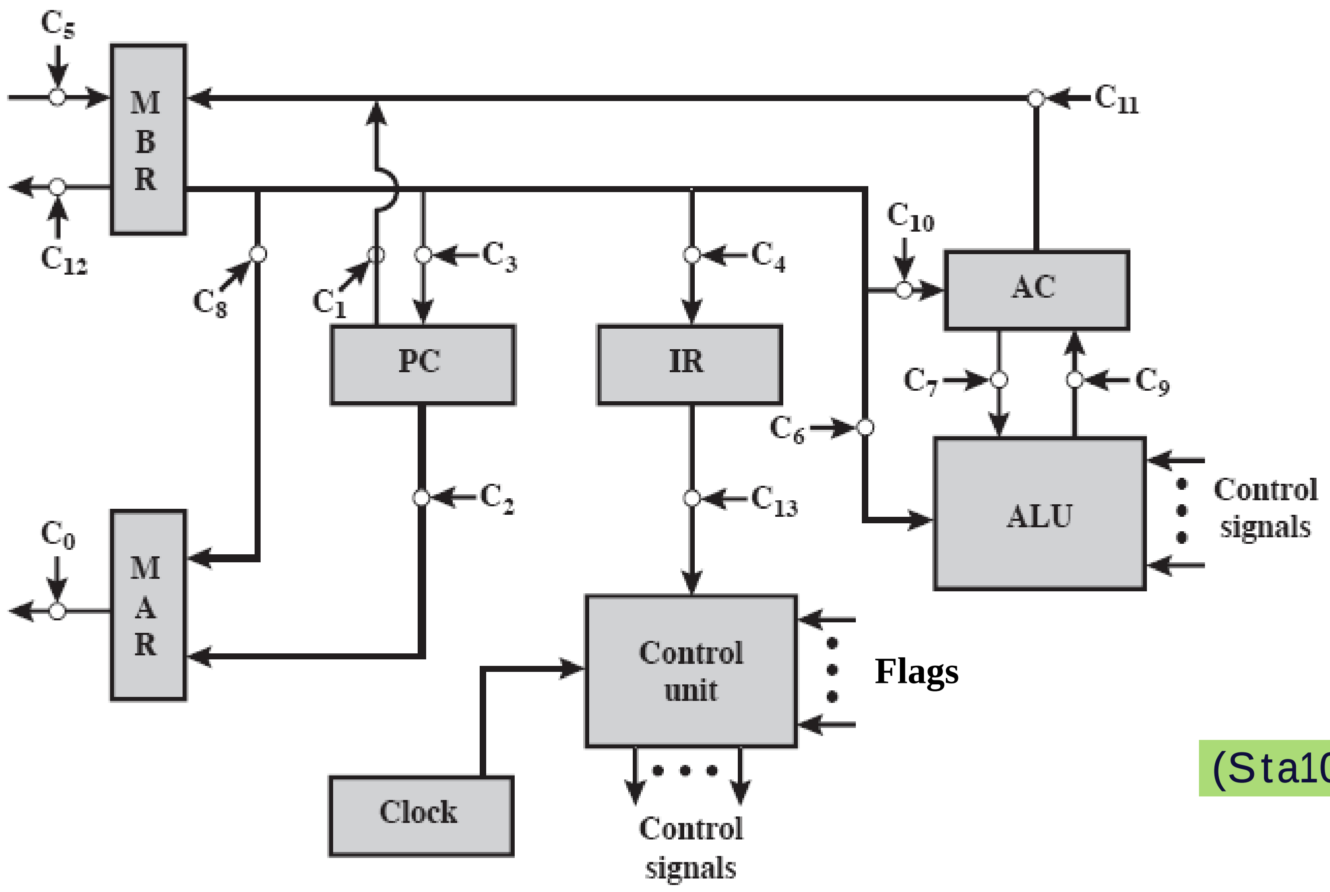
- Functionality of Control Unit can be presented as state-machine
  - State: What stage of the instruction cycle is going on in CPU
  - Substate: timing based, group of micro-operations executed parallel in one (sub)cycle
- Substate control signals are based on
  - (sub)state itself
  - Fields of IR-register (opcode, operands)
  - Previous results (flags)
    = Execution
- New state based on previous state and flags
  - Also external interrupts effect the new state
    = Sequencing



Discussion?

Control signals

- Micro-operation ⇨ CU emits a set of control signals
- Example: processor with single accumulator



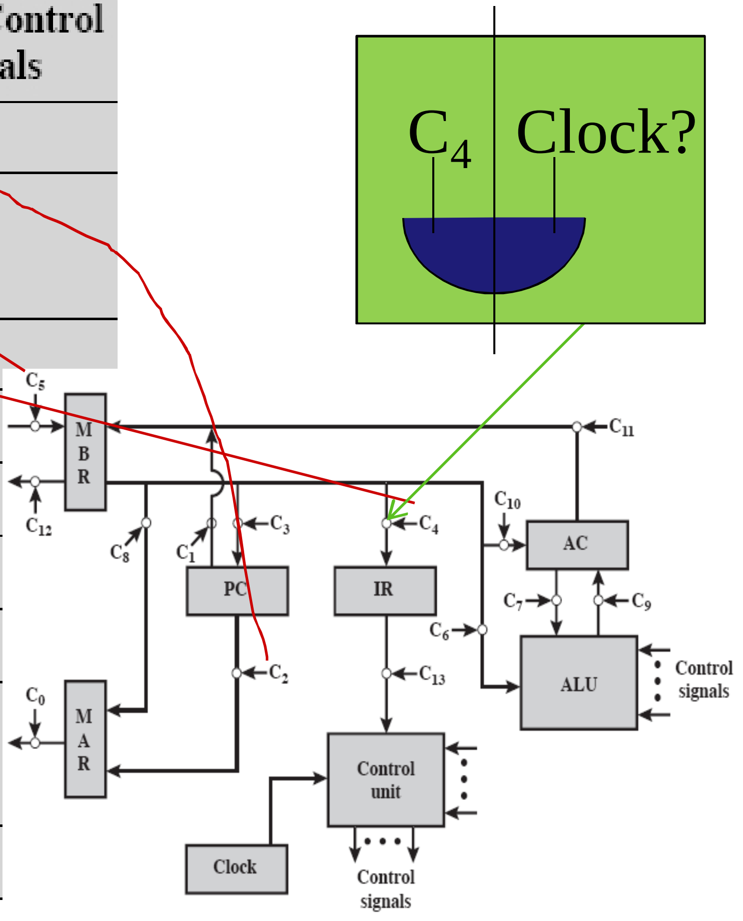
(Sta10 Fig 15.5)

Control Signals and Micro-Operations

Micro-operations	Timing	Active Control Signals
Fetch:	t_1: MAR ← (PC)	C_2
	t_2: MBR ← Memory PC ← (PC) + 1	C_5, C_R ??
	t_3: IR ← (MBR)	C_4
Indirect:	t_1: MAR ← (IR(Address))	C_8
	t_2: MBR ← Memory	C_5, C_R
	t_3: IR(Address) ← (MBR(Address))	C_4
Interrupt:	t_1: MBR ← (PC)	C_1
	t_2: MAR ← Save-address PC ← Routine-address	??
	t_3: Memory ← (MBR)	C_{12}, C_W

C_R = Read control signal to system bus.
C_W = Write control signal to system bus.

(Sta10 Table 15.1)

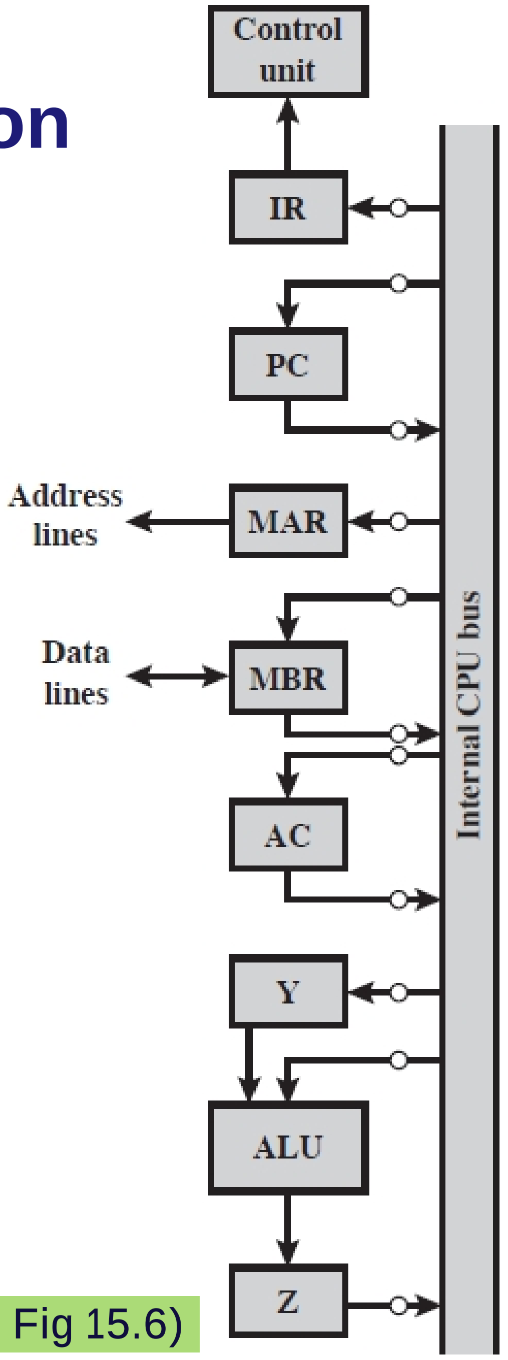


(Sta10 Fig 15.5)

Internal Processor Organization

- Fig 15.5 too complex wiring for implementation?
- Use internal processor bus to connect the components
- ALU usually has temporary registers Y and Z

ADD I:
t1: MAR ← IR.address
t2: MBR ← MEM[MAR]
t3: Y ← MBR
t4: Z ← AC + Y
t5: AC ← Z



(Sta10 Fig 15.6)

Hardwired implementation

(*Langoitettu ohjaus*)

Hardwired control unit (*Langoitettu ohjausyksikkö*)

- Can be used when CU's inputs and outputs fixed
  - Functionality described using Boolean logic
  - CU implemented by one logical circuit

C5 = *"read bus to MBR"*

- Eg. $C5 = \overline{P}*\overline{Q}*T2 + \overline{P}*Q*(LDA)*T2 + ...$

Fig 15.3, 15.5 and Tbl 15.1

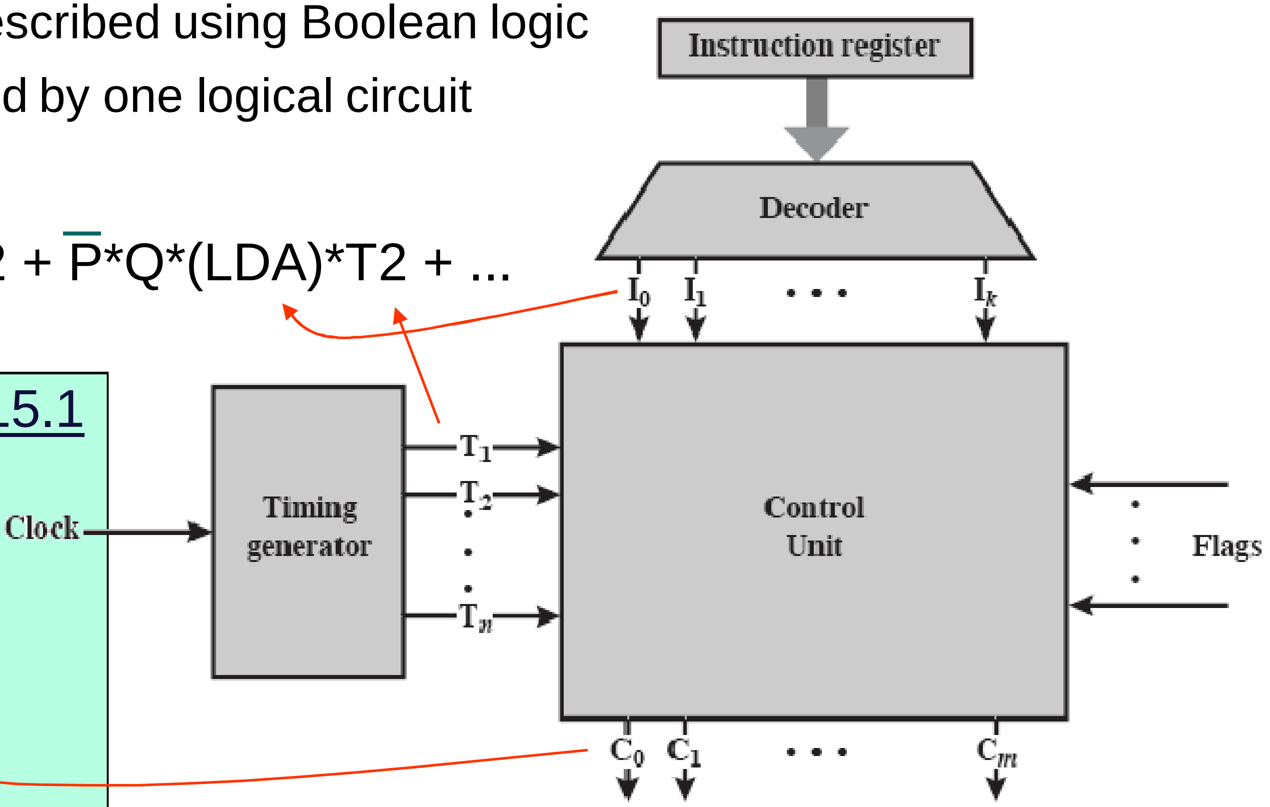
ICC - bits P and Q

PQ = 00 Fetch Cycle

PQ = 01 Indirect Cycle

PQ = 10 Execute Cycle

PQ = 11 Interrupt Cycle

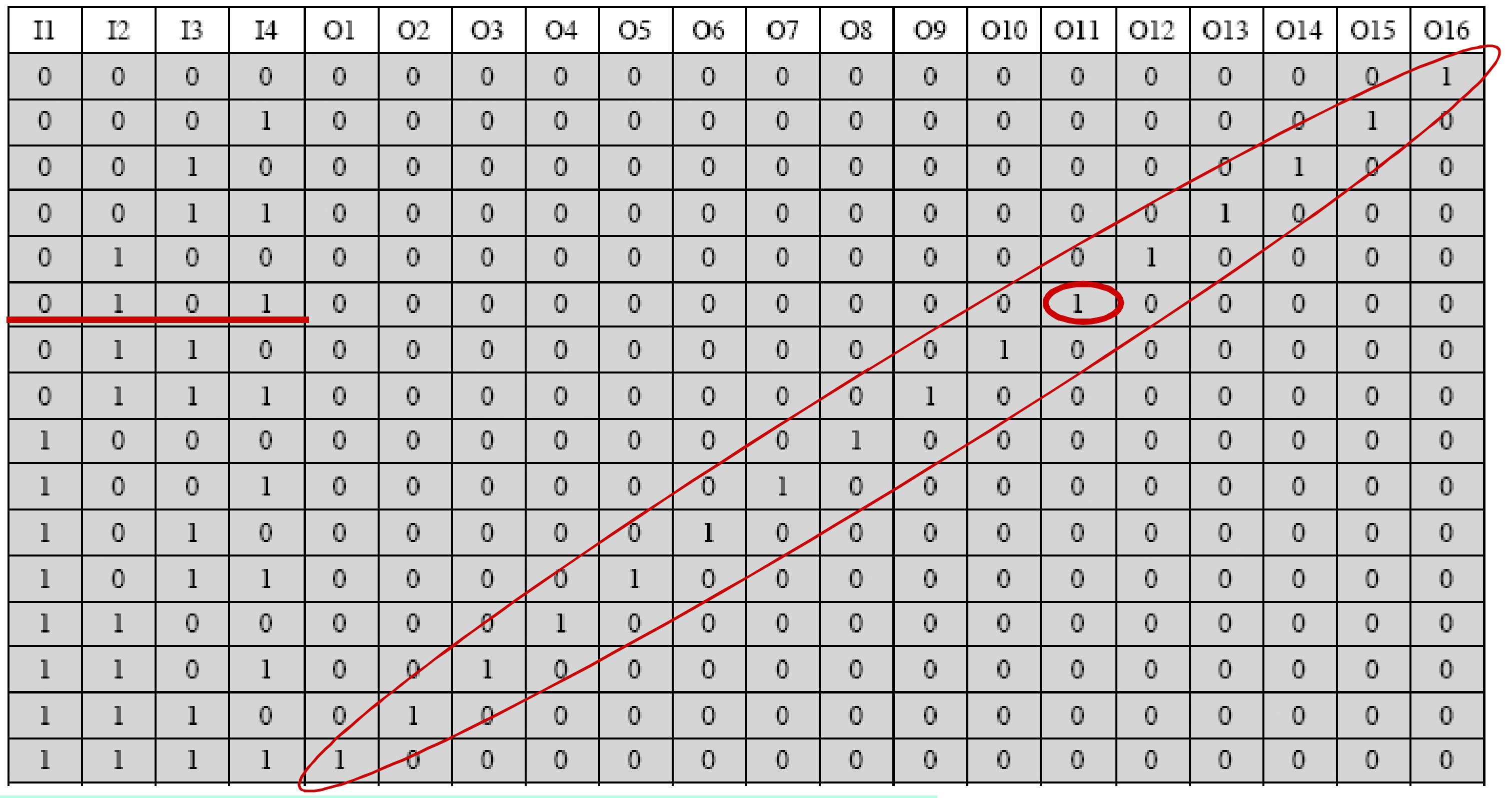


(Sta10 Fig 15.10)

Hardwired Control Unit

(Sta10 Table 15.3)

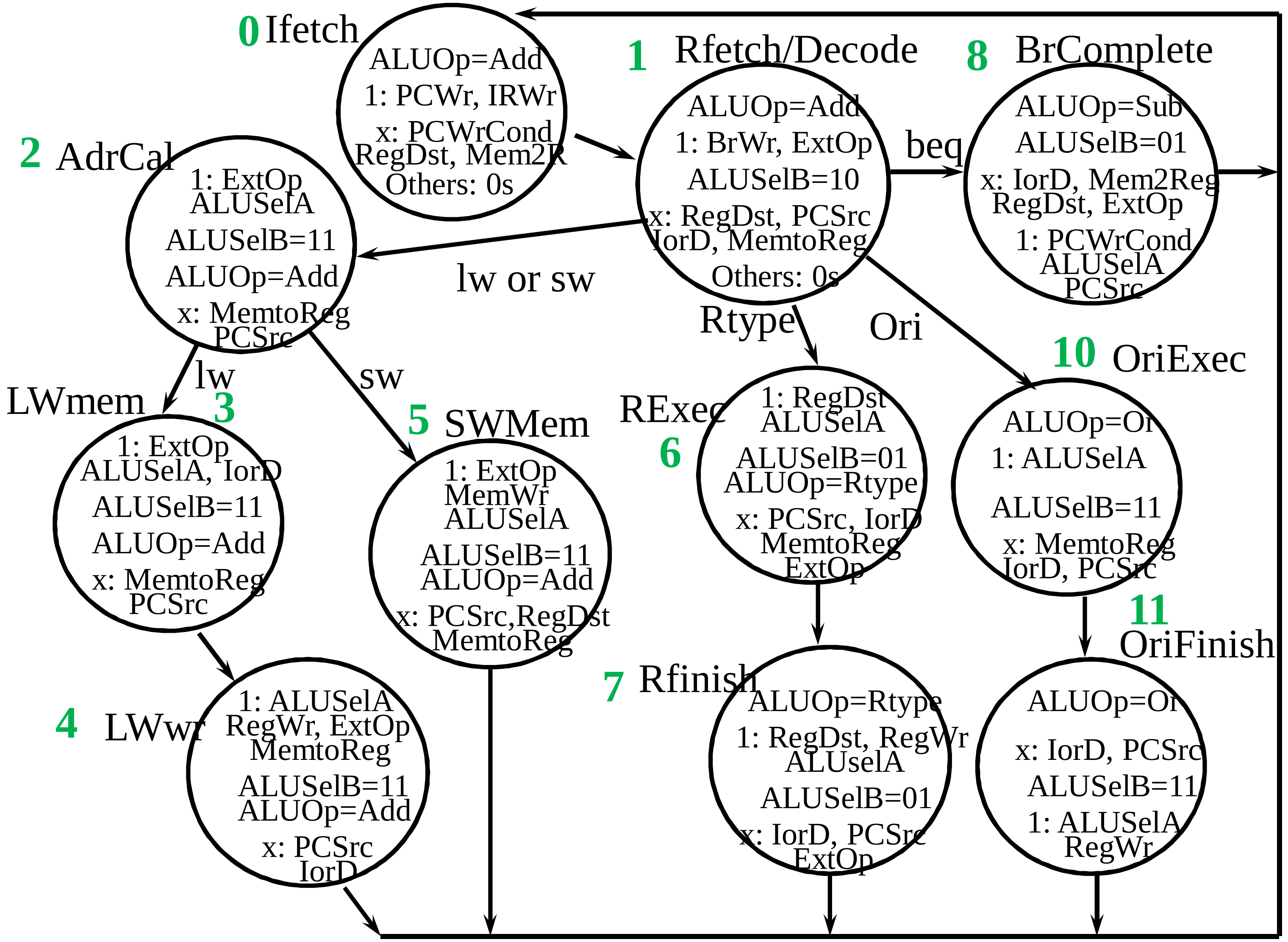
- Opcode decoder (4-to-16)
  - 4-bit instruction code as input to CU
  - Only one signal active at any given stage



I1	I2	I3	I4	O1	O2	O3	O4	O5	O6	O7	O8	O9	O10	O11	O12	O13	O14	O15	O16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

C5: opcode = 5 (bits I1, I2, I3, I4) $\longrightarrow$ signal O11 is true (1)

Finite State Diagram



State transitions

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	-> State 0
____________________	-> State1
____________________	-> State 2
____________________	-> State 3
____________________	-> State 4
State 2 & op = SW	-> State 5
____________________	-> State 6
State 6	-> State 7
____________________	-> State 8
State 3 & op = JMP	-> State 9
____________________	-> State 10
State 10	-> State 11

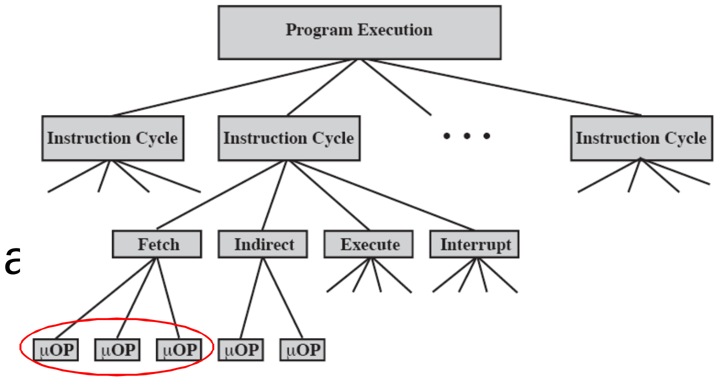
Hardwired Control Summary

- Control signal generation in hardware is fast
- Weaknesses
  - CU difficult to design
    - Circuit can become large and complex
  - CU difficult to modify and change
    - Design and ’minimizing’ must be done again after every change
- RISC-philosophy makes it a bit easier
  - Simple instruction set makes the design and implementation easier

Microprogrammed Control (*Mikro-ohjelmoitu ohjaus*)

Microprogrammed Control (*Mikro-ohjelmoitu ohjaus*)

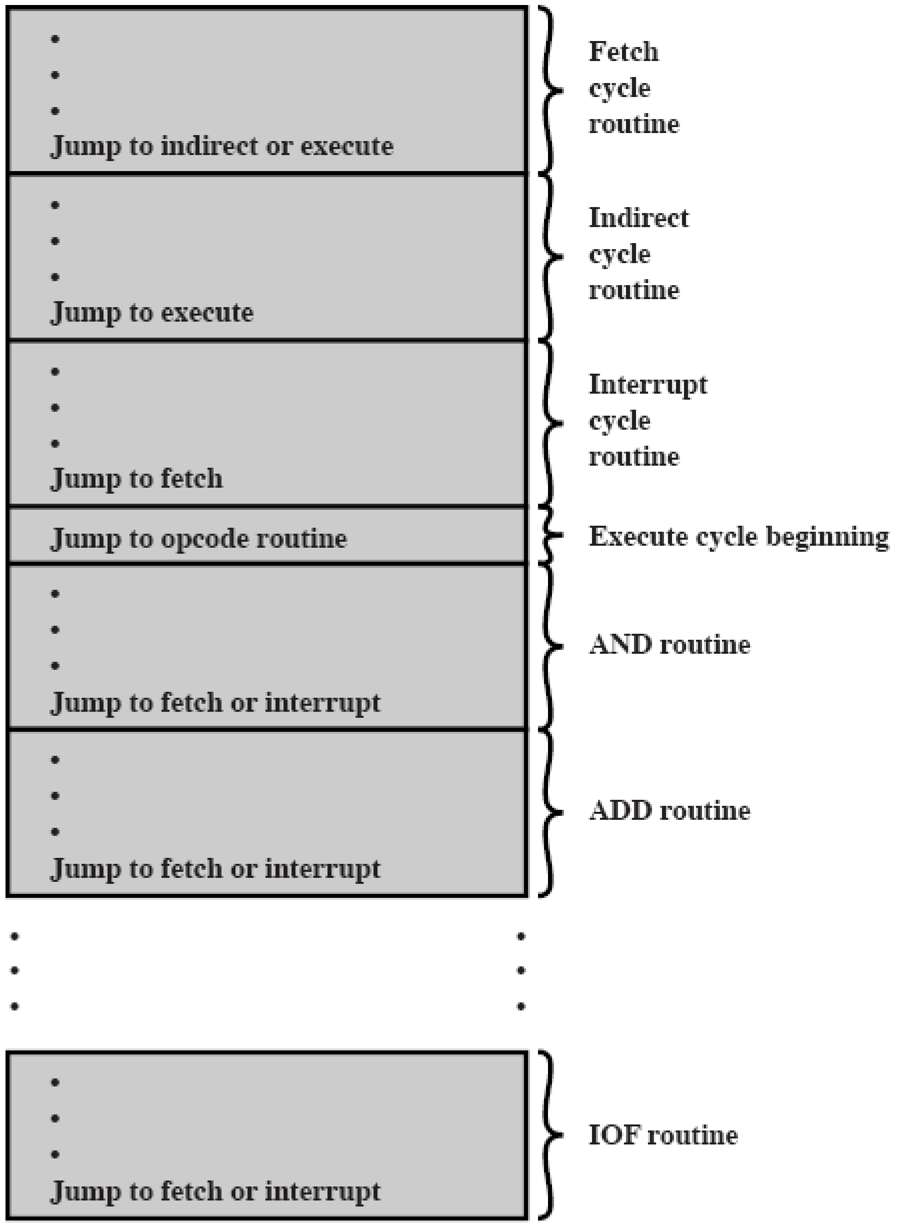
- Idea 1951: Wilkes Microprogrammed Control (Maurice Wilkes)
- Execution Engine
  - Execution of one machine instruction is done by executing a sequence of microinstructions (micro-operations)
  - Executes each microinstruction by generating the control signals indicated by the instruction
- Micro-operations stored in <u>control memory as microinstructions</u>
  - Firmware *(laiteohjelmisto*)
- Each microinstruction has two parts
  - What is done during the coming clock cycle?
    - Microinstruction indicates the control signa
    - Deliver the control signals to circuits
  - What/where is the next microinstruction?
    - Assumption: next microinstruction from next location
    - Microinstruction can contain the address of next microinstruction!



(Sta10 Fig 15.11)

Microinstructions

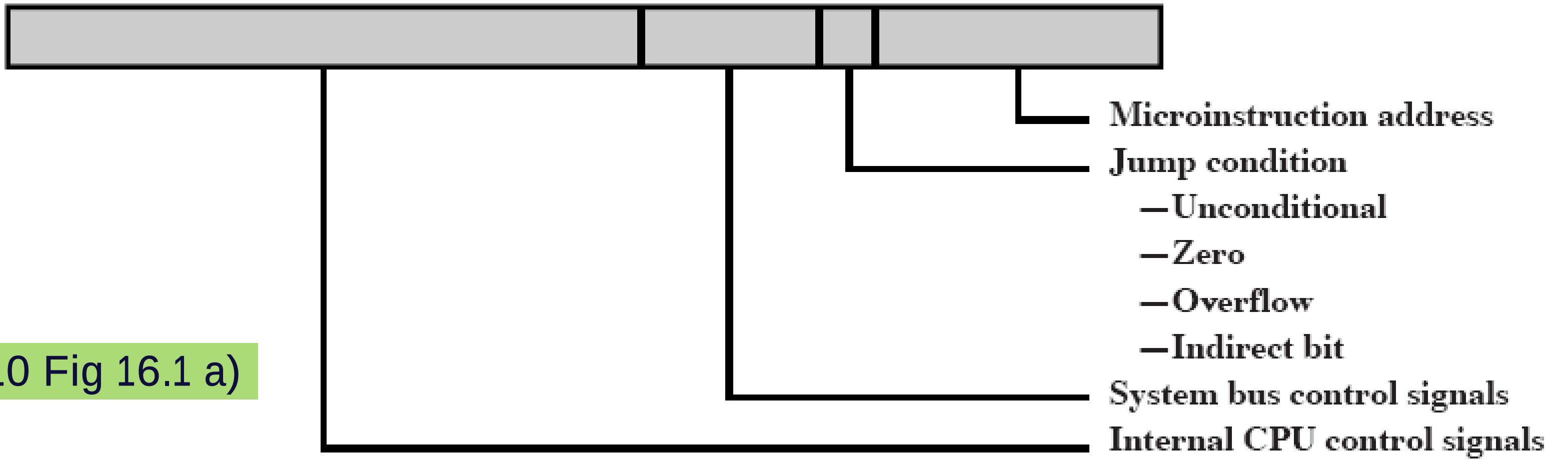
- Each stage in instruction execution cycle is represented by a sequence of microinstructions that are executed during the cycle in that stage
- E.g. in ROM
  - Microprogram or firmware



(Sta10 Fig 16.2)

Horizontal microinstruction

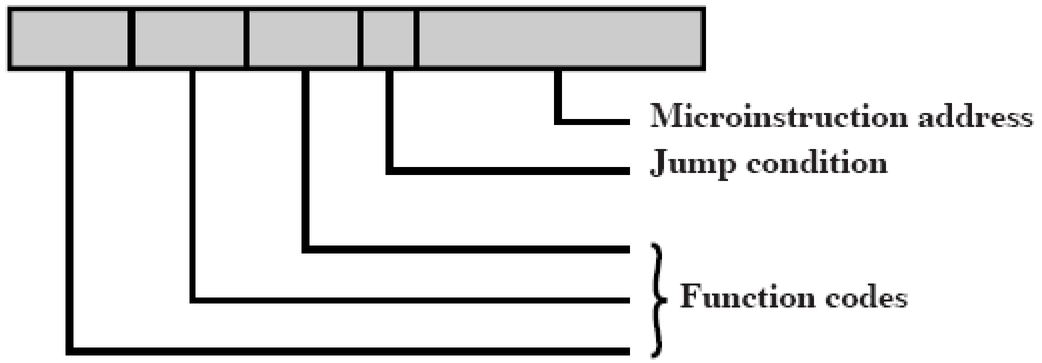
- All possible control signals are represented in a bit vector of each microinstruction
  - One bit for each signal (1=generate, 0=do not generate)
  - Long instructions if plenty of signals used
- Each microinstruction is a conditional branch
  - What status bit(s) checked
  - Address of the next microinstruction



(Sta10 Fig 16.1 a)

Vertical Microinstruction

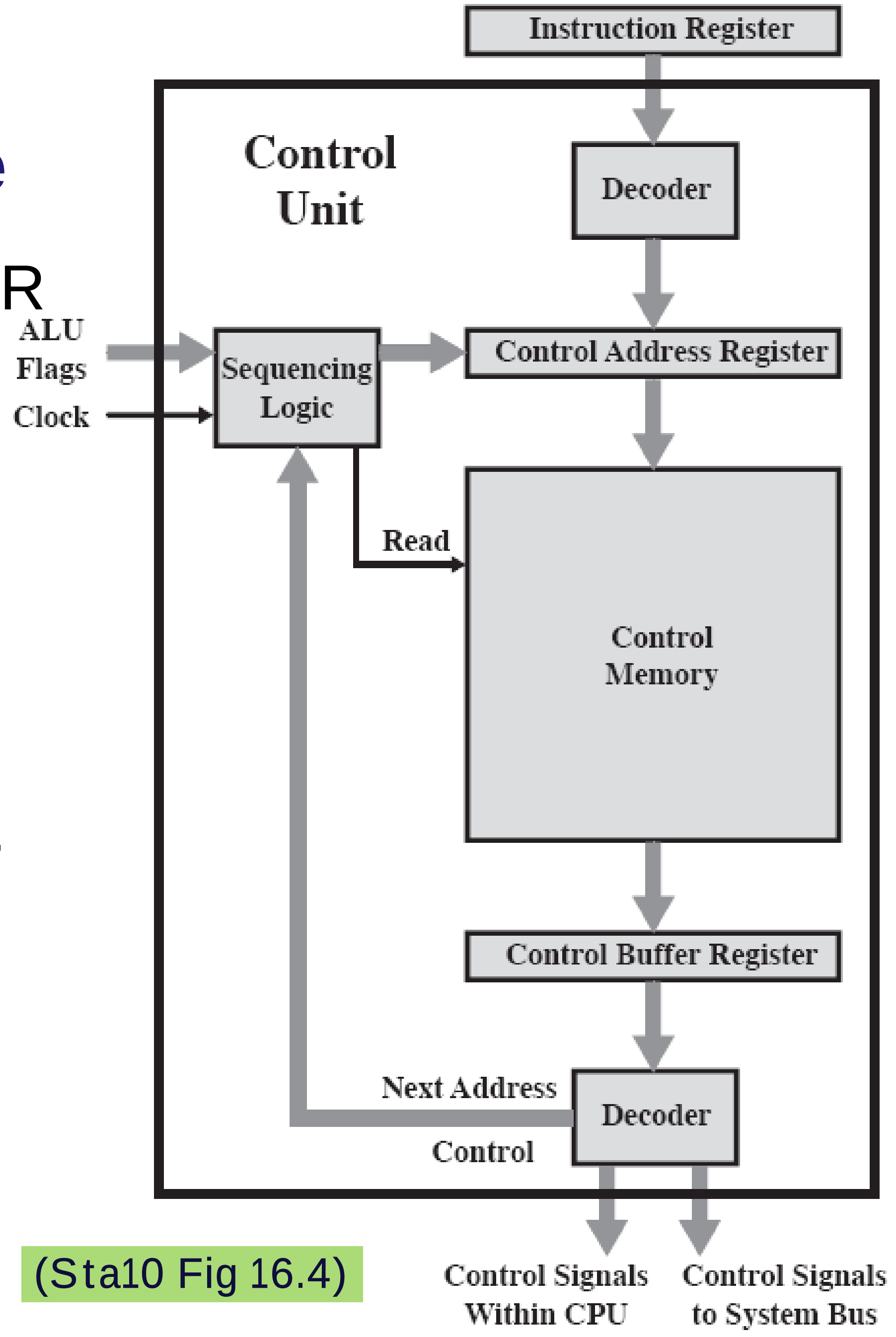
- Control signals coded to number (function)
- Decode back to control signals during execution
- Shorter instructions, but decoding takes time
  - Gate delay?
- Each microinstruction is conditional branch (as with horizontal instructions)



(Sta10 Fig 16.1 b)

Microinstruction Execution Engine

- Control Address Register, CAR
  - Which microinstruction next?
  - ~ instr. pointer, "MiPC"
- Control memory
  - Microinstructions
    - fetch, indirect,execute,interrupt
- Control Buffer Register, CBR
  - Register for executing microinstr.
  - ~ instr. register, "MiIR"
  - Generate the signals to circuits
    - Verticals through decoder
- Sequencing Logic
  - Next address to CAR

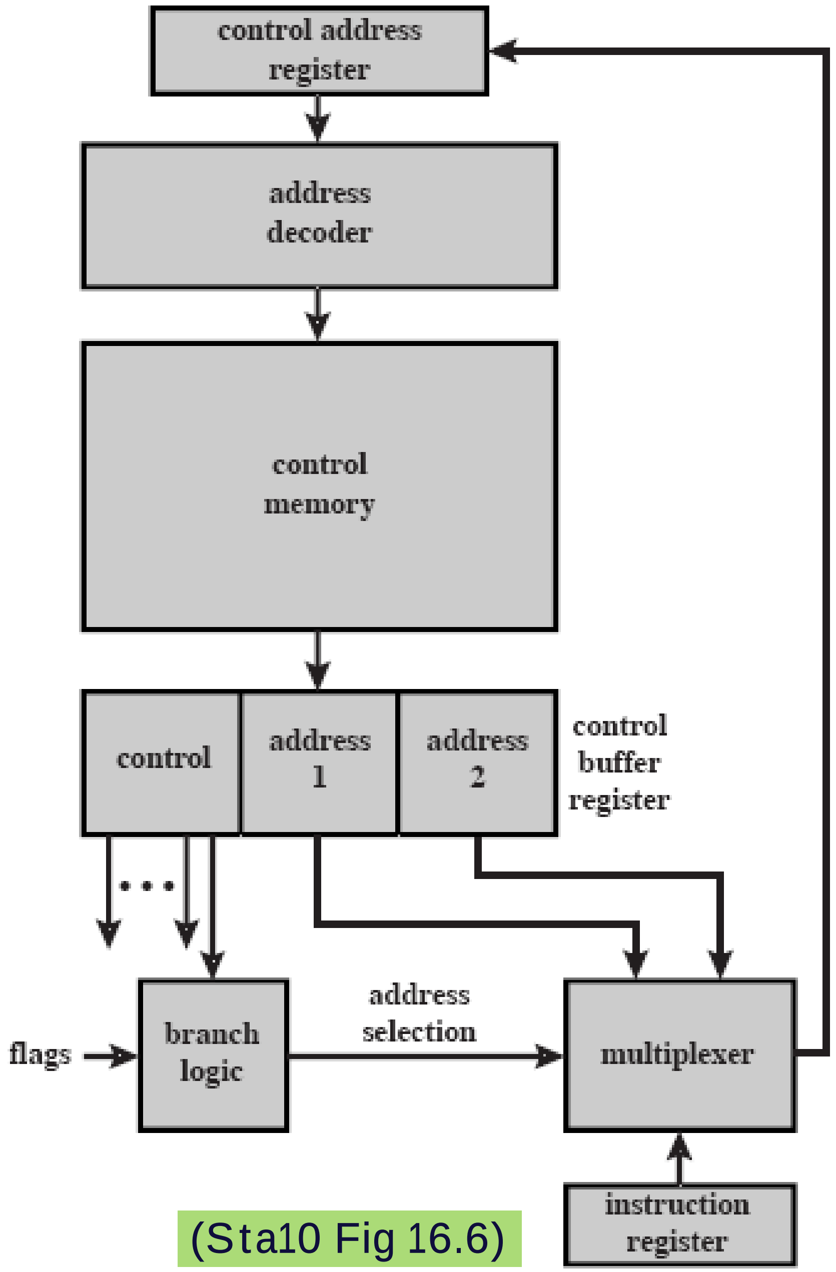


(Sta10 Fig 16.4)

Which Microinstruction Next?

a) Explicit

- Each instruction has 2 addresses
  - With the conditions flags that are checked for branching
  - Next instruction from either address (select using the flags)
  - Often just the next location in control memory
    - Why store the address?
    - No time for addition!



(Sta10 Fig 16.6)

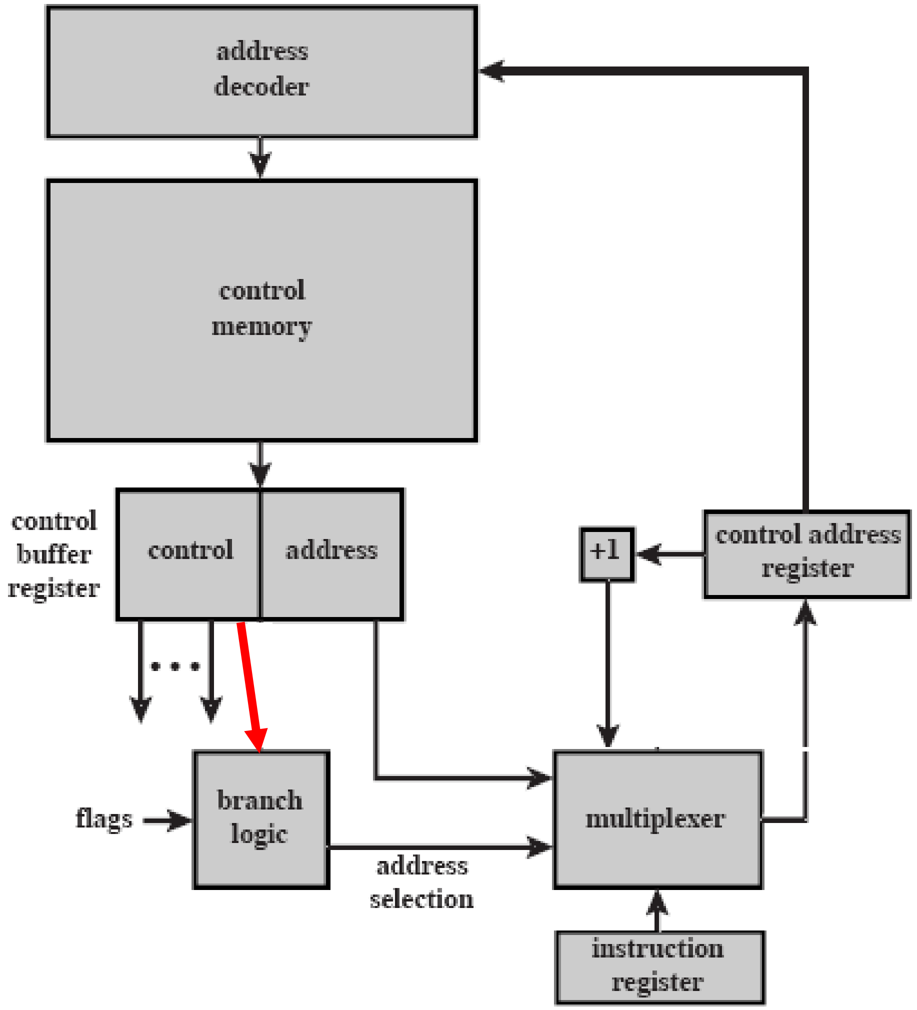
Discussion?

Which Microinstruction Next?

b) Implicit

- Assumption: next microinstruction from next location in control memory
  - Must be calculated
- μInstruction has 1 address
  - Still need condition flags
  - If condition=1,

    use the address
- Address part not always used
  - Wasted space

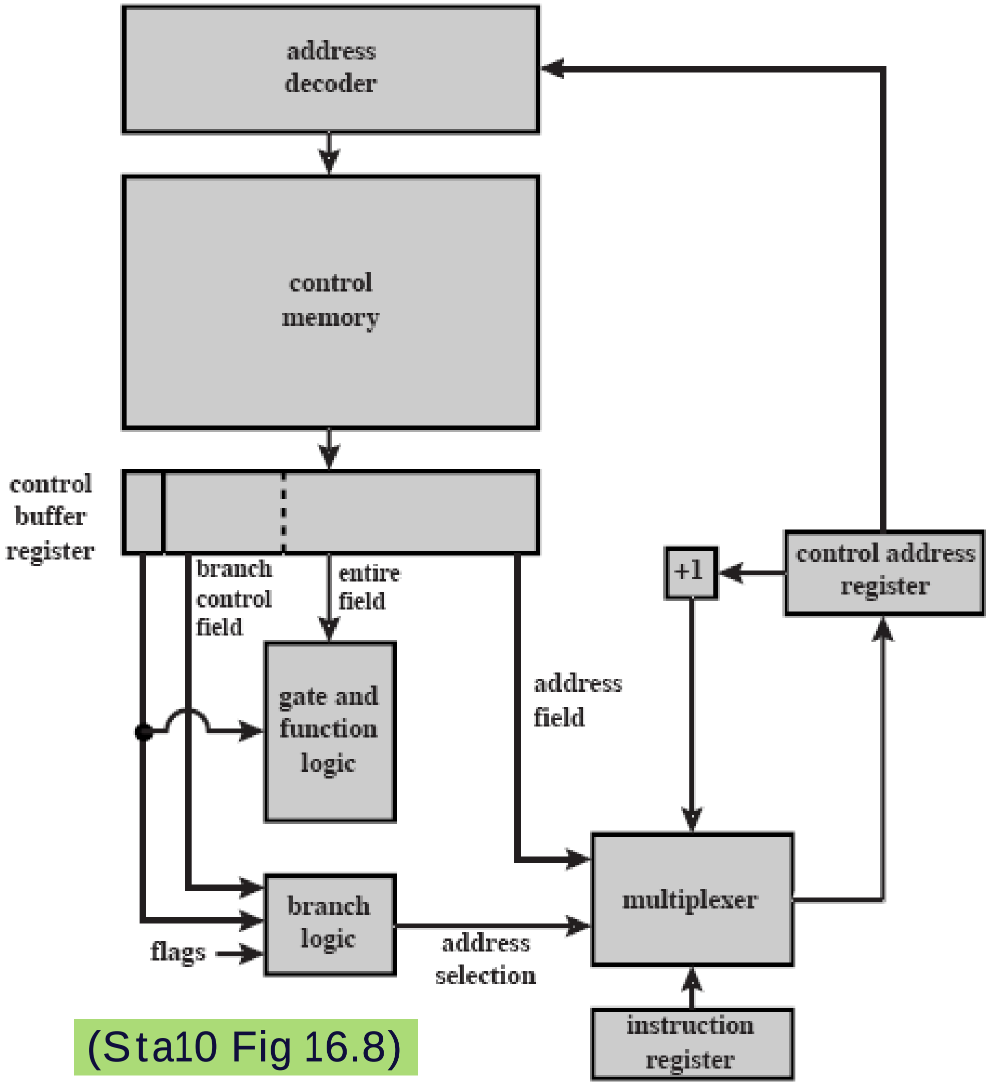


(Sta10 Fig 16.7)

Which Microinstruction Next?

c) Variable format

- Some bits interpreted in two ways
  - 1 b: Address or not
  - Only branch instructions have address
  - Branch instructions do not have control signals
  - If jump, need to execute two microinstructions instead of just one
    - Wasted time?
    - Saved space?

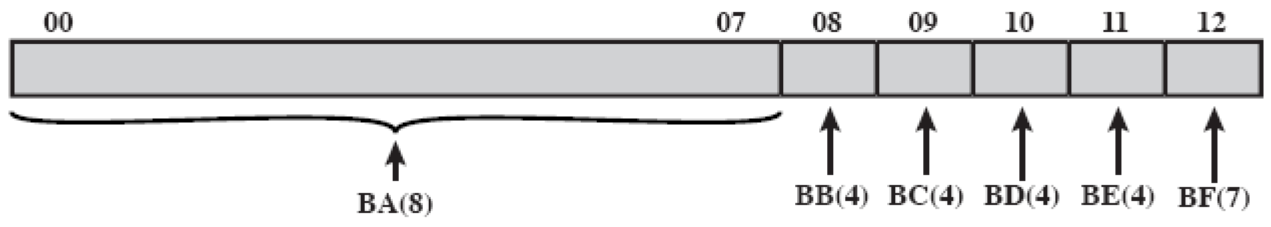


(Sta10 Fig 16.8)

Which Microinstruction Next?

d) Address generation during execution

- How to locate the correct microinstruction routine?
  - Control signals depend on the current machine instruction
- Generate first microinstruction address from op-code (mapping + combining/adding)
  - Most-significant bits of address directly from op-code
  - Least-significant bits based on the current situation (0 or 1)
  - Example: IBM 3033 Control Address Register (CAR), 13 bit address
    - Op-code gives 8 bits -> each sequence 32 micro-instr.
    - rest 5 bits based on the certain status bits



(Sta10 Fig 16.9)

Which Microinstruction Next?

e) Subroutines and residual control

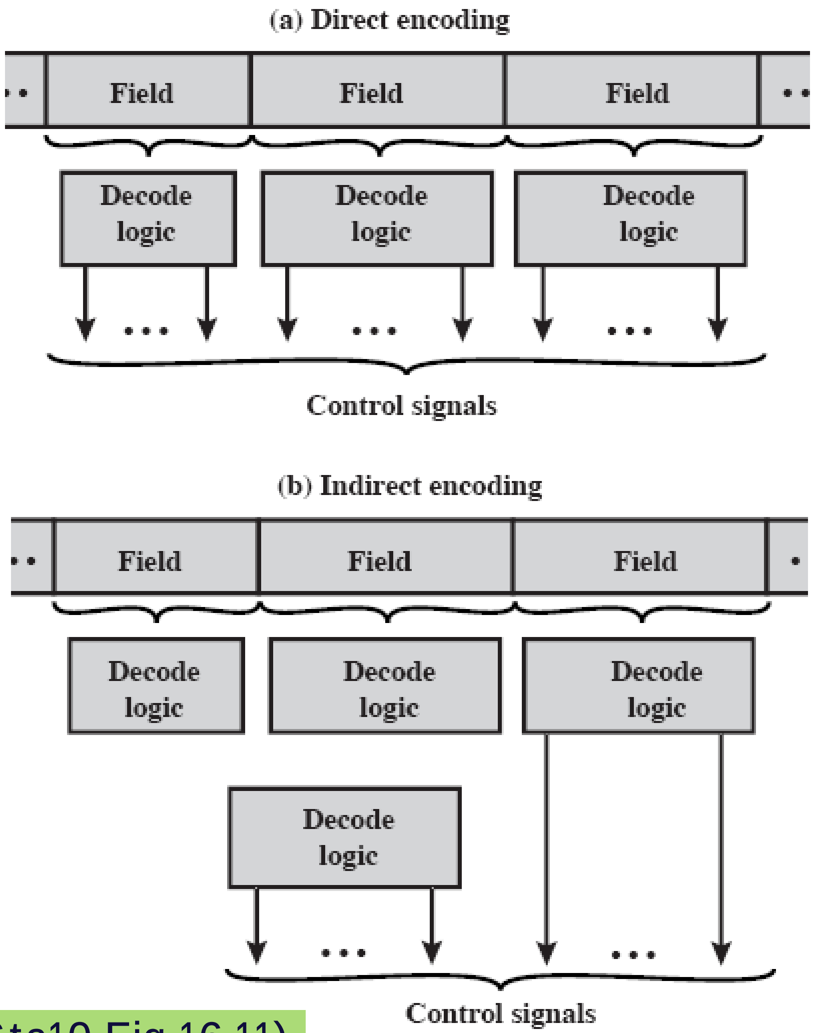
- Microinstruction can set a special <u>return register</u> with 'return address'
  - No context, just one return allowed (one-level only)
  - No nested structure
  - Example: LSI-11, 22 bit microinstruction
    - Control memory 2048 instructions, 11 bit address
    - OP-code determines the first microinstruction address
    - Assumption, next is CAR ← CAR+1
    - Each instruction has a bit: subroutine call or not
    - Call:
      - Store return address (only the latest one available)
      - Jump to the routine (address in the instruction)
    - Return: jump to address in return register

Microinstruction Coding

- Horizontal or Vertical?
  - Horizontal: fast interpretation
  - Vertical: less bits, smaller space
- Often a compromize, using mixed model
  - Microinstruction split to fields, each field is used for certain control signals
  - Excluding signal combinations can be coded in the same field
    - NOT: Reg source and destination, two sources – one dest
  - Coding decoded to control signals during execution
    - One field can control decoding of other fields!
- Several shorter coded fields easier for implementation than one long field
  - Several simple decoders

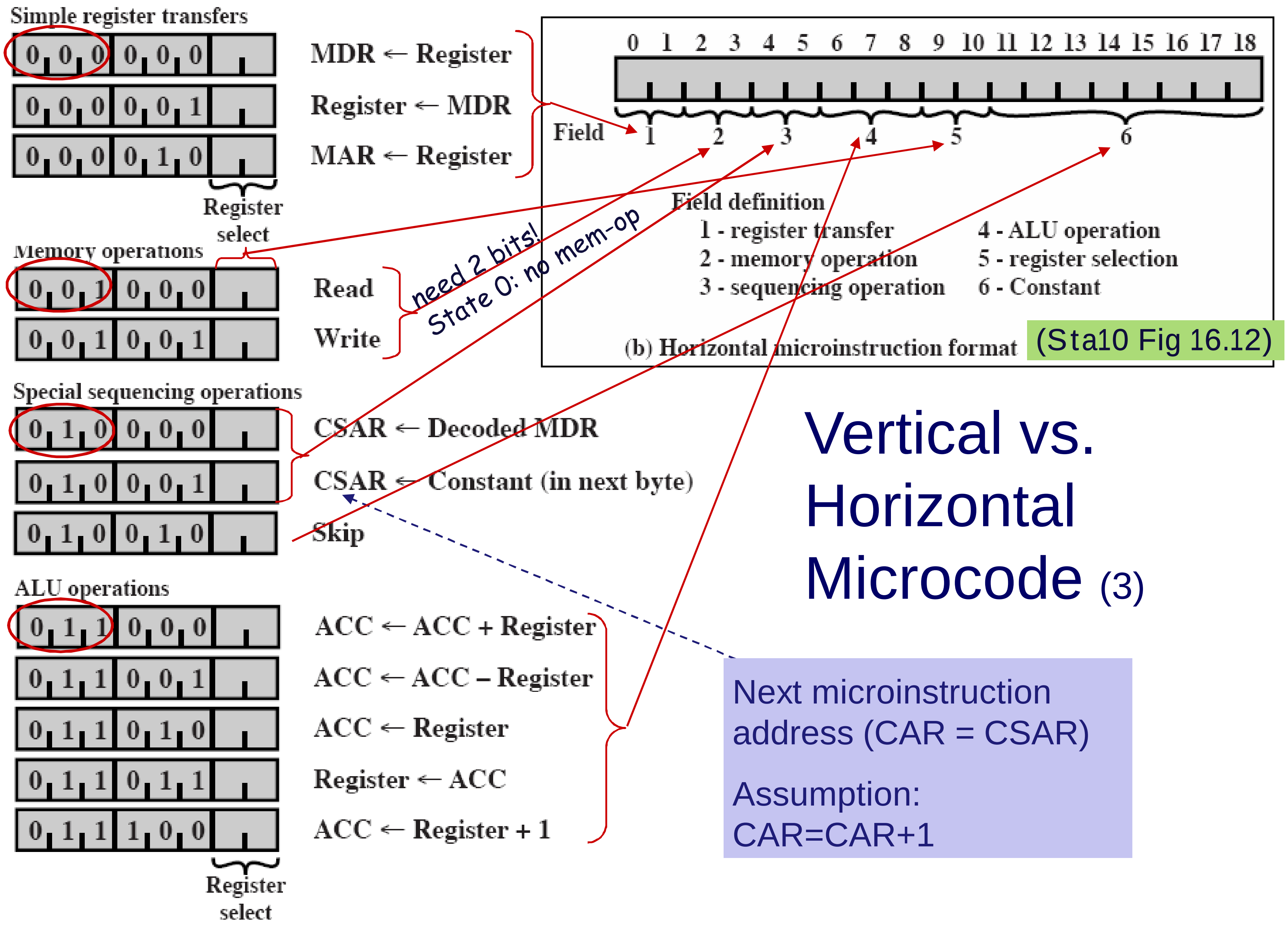
Microinstruction Coding

- Functional encoding (*toiminnoittain*)
  - Each field controls one specific action (e.g., load)
    - Load from accumulator
    - Load from memory
    - Load from ...
- Resource encoding (*resursseittain*)
  - Each field controls specific resource (e.g. accumulator)
    - Load from accumulator
    - Store to accumulator
    - Add to accumulator
    - … accumulator



(Sta10 Fig 16.11)

Vertical vs. Horizontal Microcode (3)



Simple register transfers
0 0 0 0 0 0 MDR ← Register
0 0 0 0 0 1 Register ← MDR
0 0 0 0 1 0 MAR ← Register
Register select
Memory operations
0 0 1 0 0 0 Read
0 0 1 0 0 1 Write
Special sequencing operations
0 1 0 0 0 0 CSAR ← Decoded MDR
0 1 0 0 0 1 CSAR ← Constant (in next byte)
0 1 0 0 1 0 Skip
ALU operations
0 1 1 0 0 0 ACC ← ACC + Register
0 1 1 0 0 1 ACC ← ACC – Register
0 1 1 0 1 0 ACC ← Register
0 1 1 0 1 1 Register ← ACC
0 1 1 1 0 0 ACC ← Register + 1
Register select
need 2 bits!
State 0: no mem-op
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
Field 1 2 3 4 5 6
Field definition
1 - register transfer
2 - memory operation
3 - sequencing operation
4 - ALU operation
5 - register selection
6 - Constant

(a) Vertical microinstruction format (by resource)

(b) Horizontal microinstruction format (Sta10 Fig 16.12)

Next microinstruction address (CAR = CSAR)

Assumption: CAR=CAR+1

Why microprogrammed control?

- … even when its slower than hardwired control
- Design is simple and flexible
  - Modifications (e.g. expansion of instruction set) can be added very late in the design phase
  - Old hardware can be updated by just changing control memory
    - Whole control unit chip in older machines
  - There exists (existed?) development environments for microprograms
- Backward compatibility
  - Old instruction set can be used easily
  - Just add new microprograms for new machine instructions
- Generality
  - One hardware, several different instruction sets
  - One instruction set, several different organizations

Control Summary

- Control signals
- Hardwired control
- Microprogrammed control?
  - Control memory, control address, control buffer
  - Horizontal vs. vertical microprogrammed control?
  - How do you find the next microinstruction?
  - LSI-11 example

Review Questions / Kertauskysymyksiä

- Hardwired vs. microprogrammed control?
- How to determine the address of microinstruction?
- What is the purpose of control memory?
- Horizontal vs. vertical microinstruction?
- Compare microprogram execution to machine language fetch-execute cycle.
- Microprogrammed vs. hardwired?