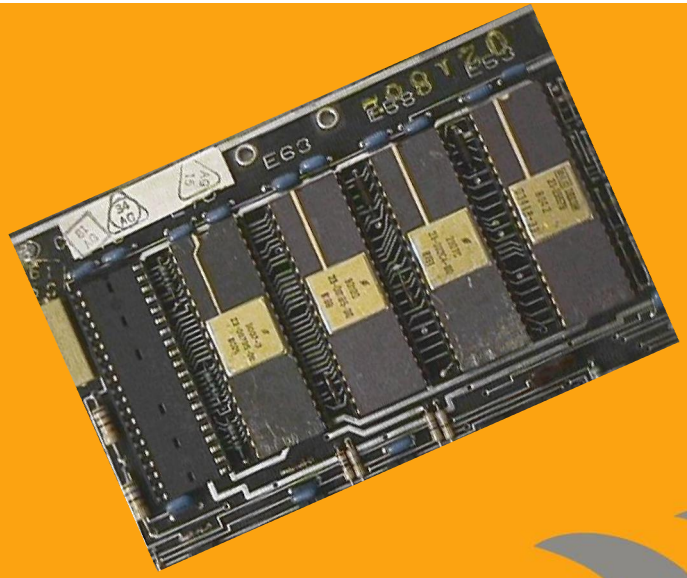
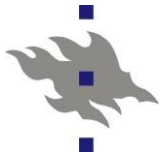
HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI



# Control Unit (*Ohjausyksikkö*)

**Ch 16-17 [Sta06]**

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



# What is control?

Functional requirements for CPU

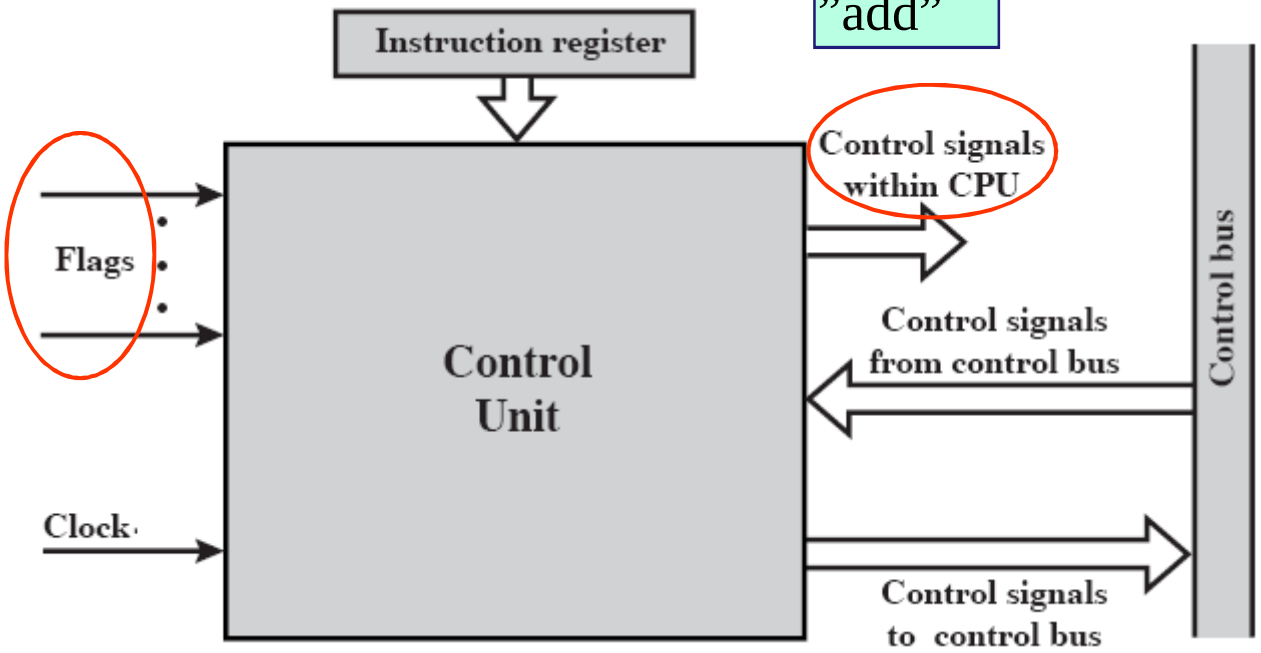
1. Operations
2. Addressing modes
3. Registers
4. I/O module interface
5. Memory module interface
6. Interrupt processing structure

- 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?
- 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

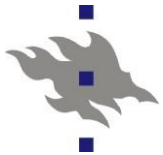
# Control signals

”read”
”write”
”add”

(Sta06 Fig 16.4)




- 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

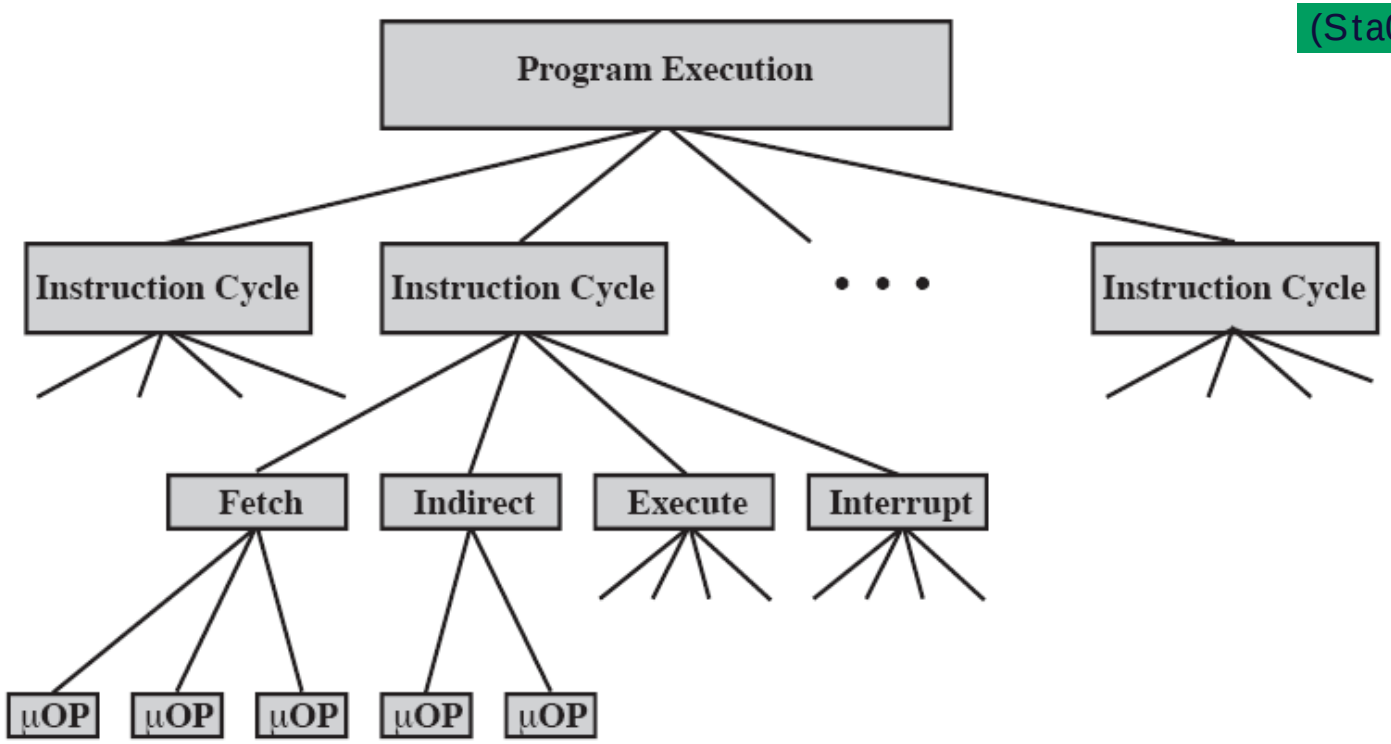
- Simple control signals that cause one very small operation (*toiminto*)
  - E.g. Bits move from reg 1 through internal bus to ALU
- Subcycle duration determined from the longest operation
- During each subcycle multiple micro-operations in action
  - Some can be done simultaneously, if in different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

t1: MAR ← PC
t2: MBR ← MEM[MAR]
PC ← PC + 1
t3: IR ← (MBR)

If implemented without ALU

# Instruction cycle (Käskysykli)

(Sta06 Fig 16.1)




- 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 (*Käskyn noutosykli*)

Example:

t1: MAR ← PC

t2: MAR ← MMU(MAR)

Control Bus ← Reserve

wait?

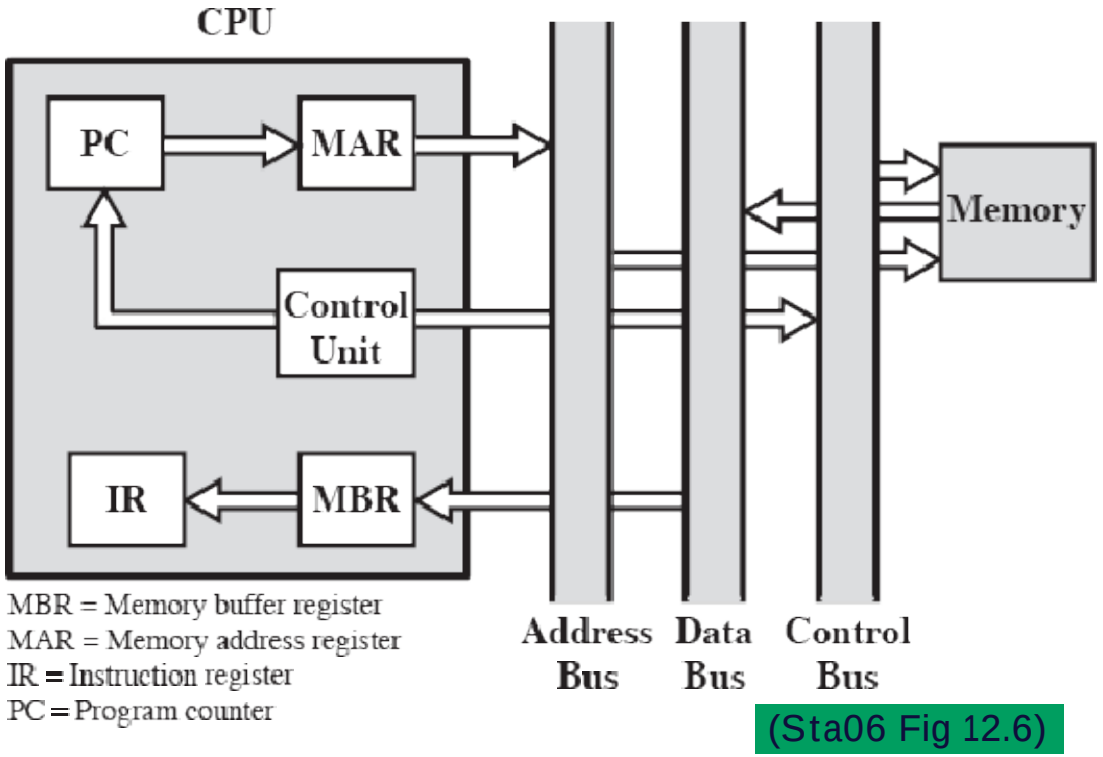
t3: Control Bus ← Read

PC ← PC + 1

t4: MBR ← MEM[MAR]

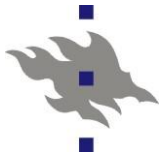
Control Bus ← Release

t5: IR ← MBR




(Sta06 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, PowerPC
  - What to same micro-operation?
  - What micro-ops parallel / sequentially?

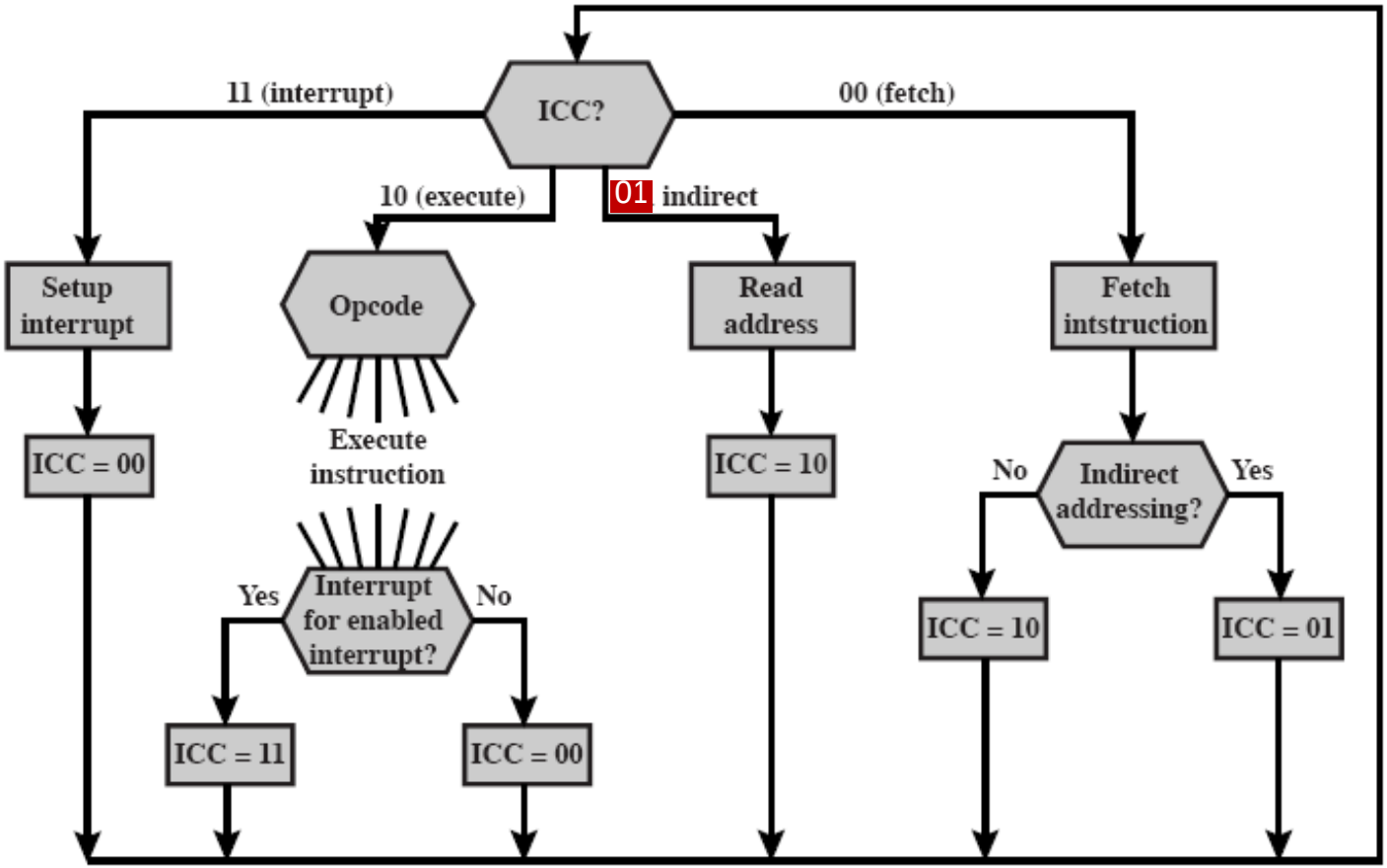
ADD r1,r2,r3:
t1: ALUin1 ← r2
t2: ALUin2 ← r3
    ALUoper ← IR.oper
t3: r1 ← ALUout
    flags ← xxx

ISZ X, Increment and Skip if zero:
t1: MAR ← IR.address
t2: MBR ← MEM[MAR]
t3: MBR ← MBR+1
t4: MEM[MAR] ← MBR
    if (MBR=0) then PC ← PC +1

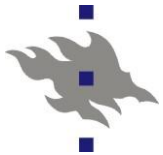
Conditional operation possible

# Instruction cycle flow chart (as state-machine?)

- ICC: Instruction Cycle Code register 's state




(Sta06 Fig 16.3)

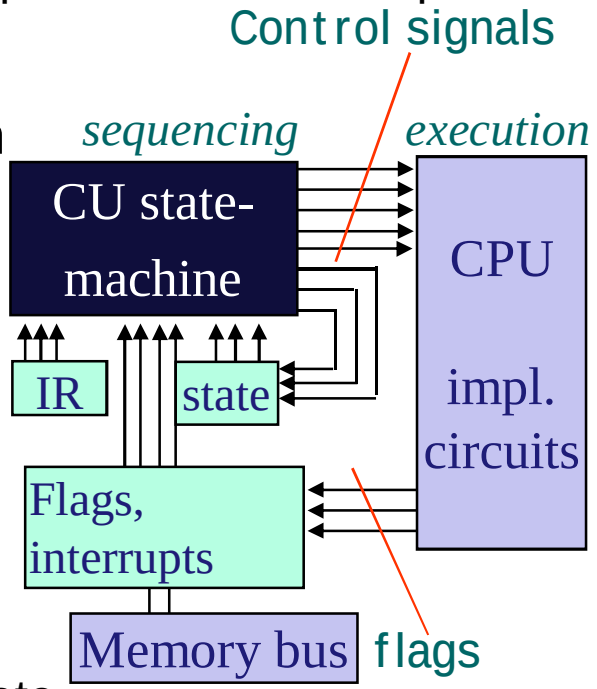


# Instruction cycle control as state-machine (*tila-automaatti)*

- 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
- Control signals of substate are based on
  - (sub)state itself
  - Fileds of IR-register (opcode, operands)
  - Previous results (flags)

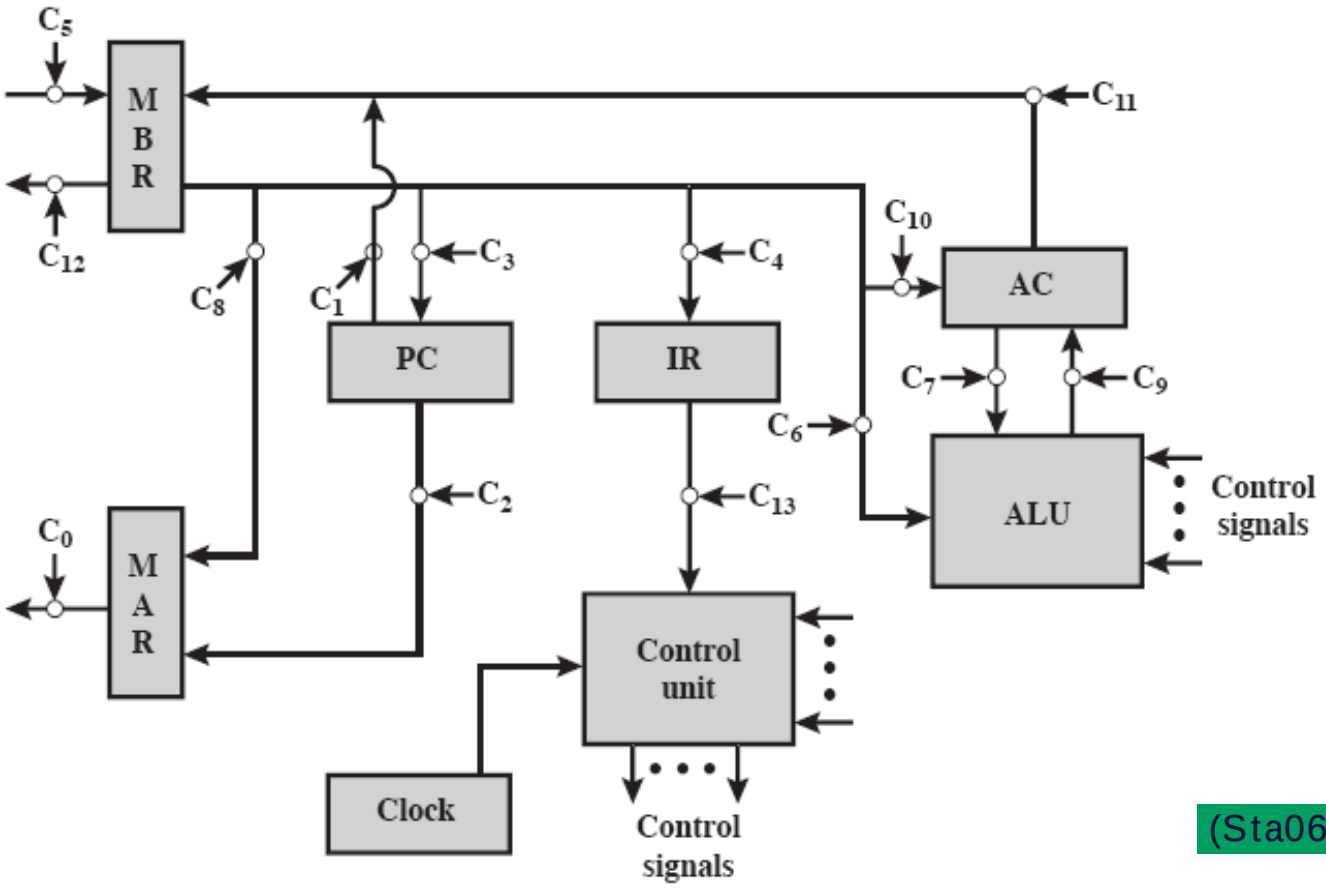
    = Execution
- New state based on previous state and flags
  - Also external interrupts effect the new state

    = Sequencing



# Control signals

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




(Sta06 Fig 16.5)

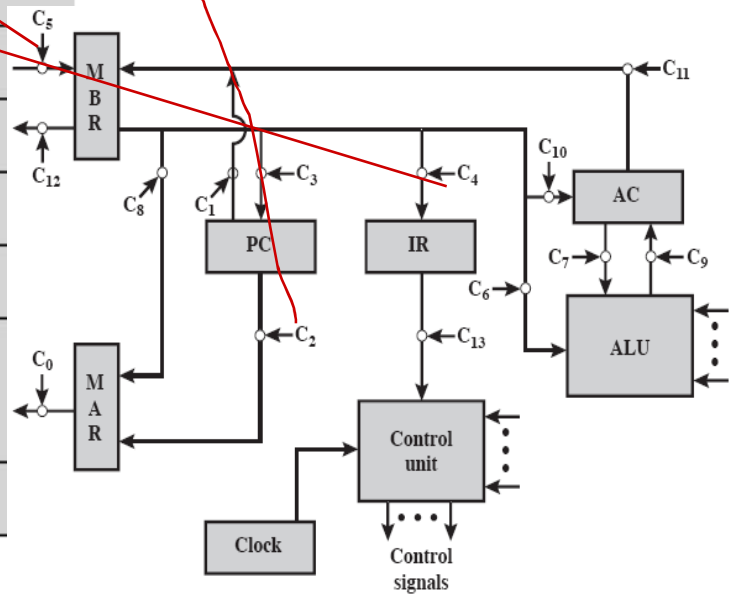
# Control signals and micro-operations

| Micro-operations | Timing | Active Control Signals |
|---|---|---|
| Fetch: | $t_1$: MAR ← (PC) | $C_2$ |
| | $t_2$: MBR ← Memory | $C_5, C_R$ |
| | PC ← (PC) + 1 | ?? |
| | $t_3$: 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.

(Sta06 Table 16.1)

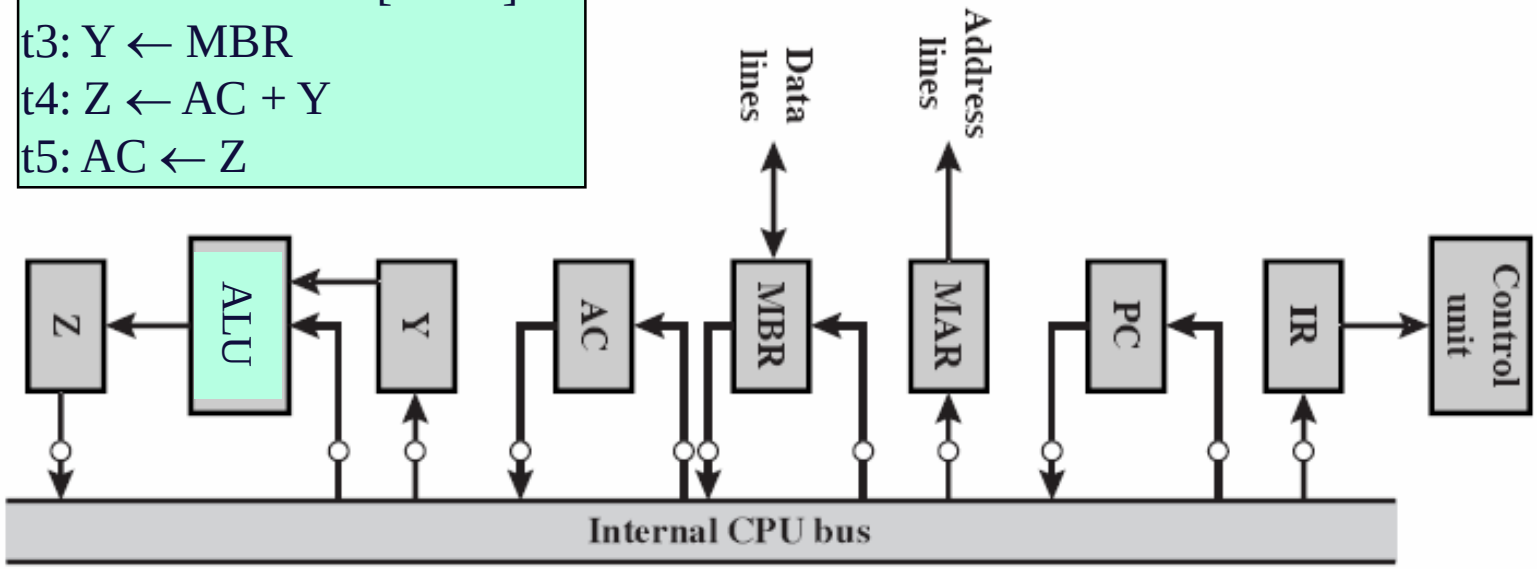
Sta06 Fig 16.5



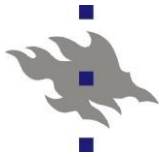
# Internal Processor Organization

- Fig 16.5 too complex 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




(Sta06 Fig 16.6)



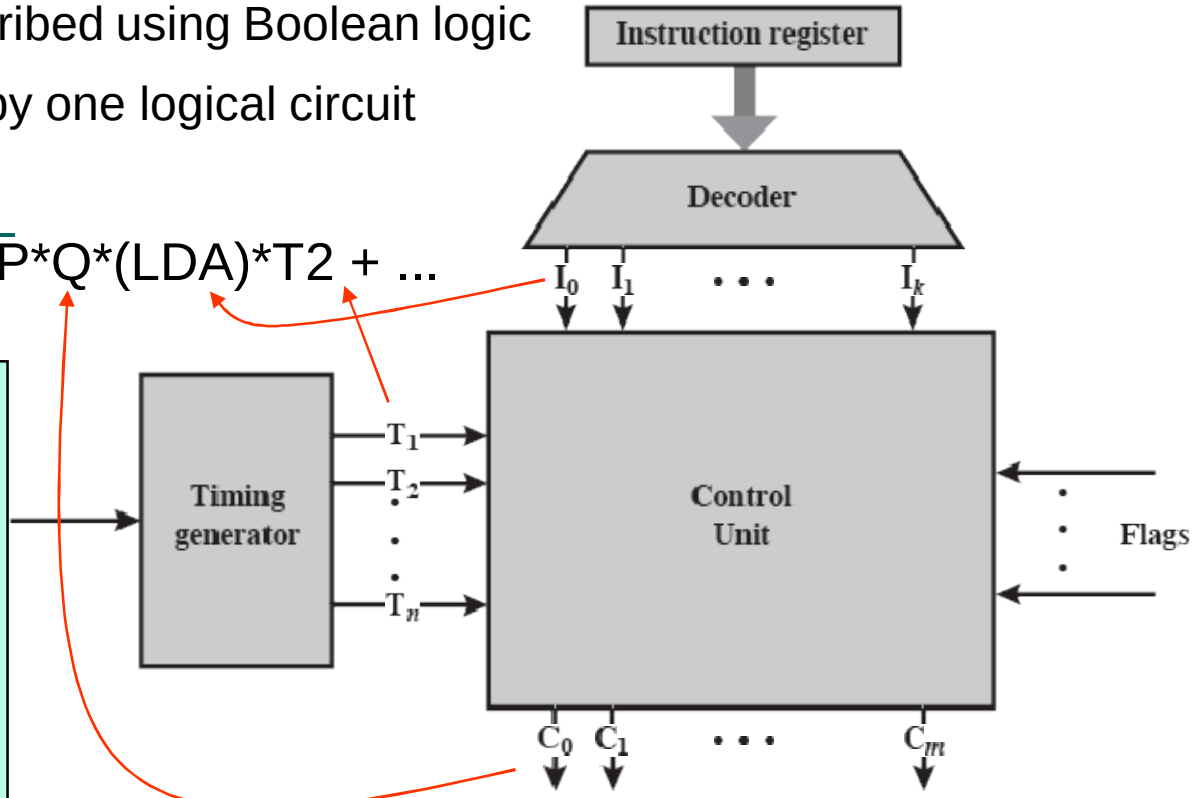


# Hardwired implementation

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




(Sta06 Fig 16.10)

Fig 16.3, 16.5 and Tbl 16.1

ICC - bits P and Q

PQ = 00 Fetch Cycle

PQ = 01 Indirect Cycle

PQ = 10 Execute Cycle

PQ = 11 Interrupt Cycle

# Hardwired control unit

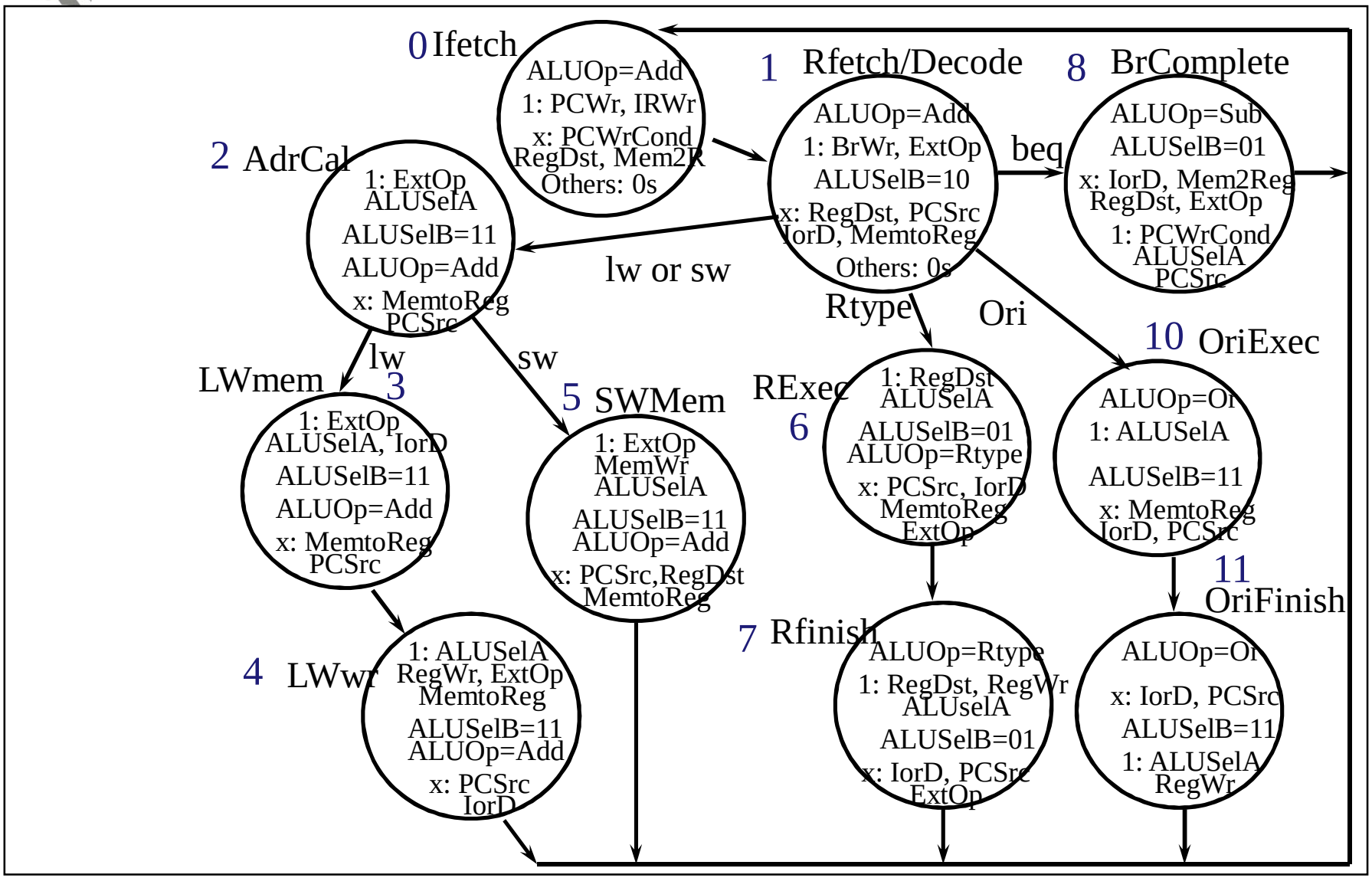
- Decoder (4-to-16)
  - 4-bit instruction code as input to CU
  - Only one signal active at any given stage

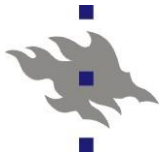
| I1 | I2 | I3 | I4 | O1 | O2 | O3 | O4 | O5 | O6 | O7 | O8 | O9 | O10 | O11 | O12 | O13 | O14 | O15 | O16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0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

opcode = 5 (bits I1, I2, I3, I4) ⟶ signal O11 is true (1)

(Sta06 Table 16.3)

# Finite State Diagram





# State transitions (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

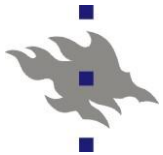
State 6 -> State 7

_____ -> State 8

State3 & op = JMP -> State 9

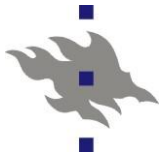
_____ -> State 10

State 10 -> State 11



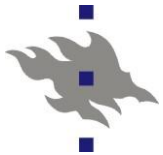
# Hardwired control

- 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
- RISC-philosophy makes it a bit easier
  - Simple instruction set makes the design and implementation easier





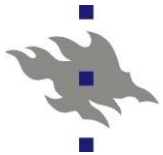
# Microprogrammed control (*Mikro-ohjelmoitu ohjaus*)



# Microprogrammed control (*Mikro-ohjelmoitu ohjaus*)

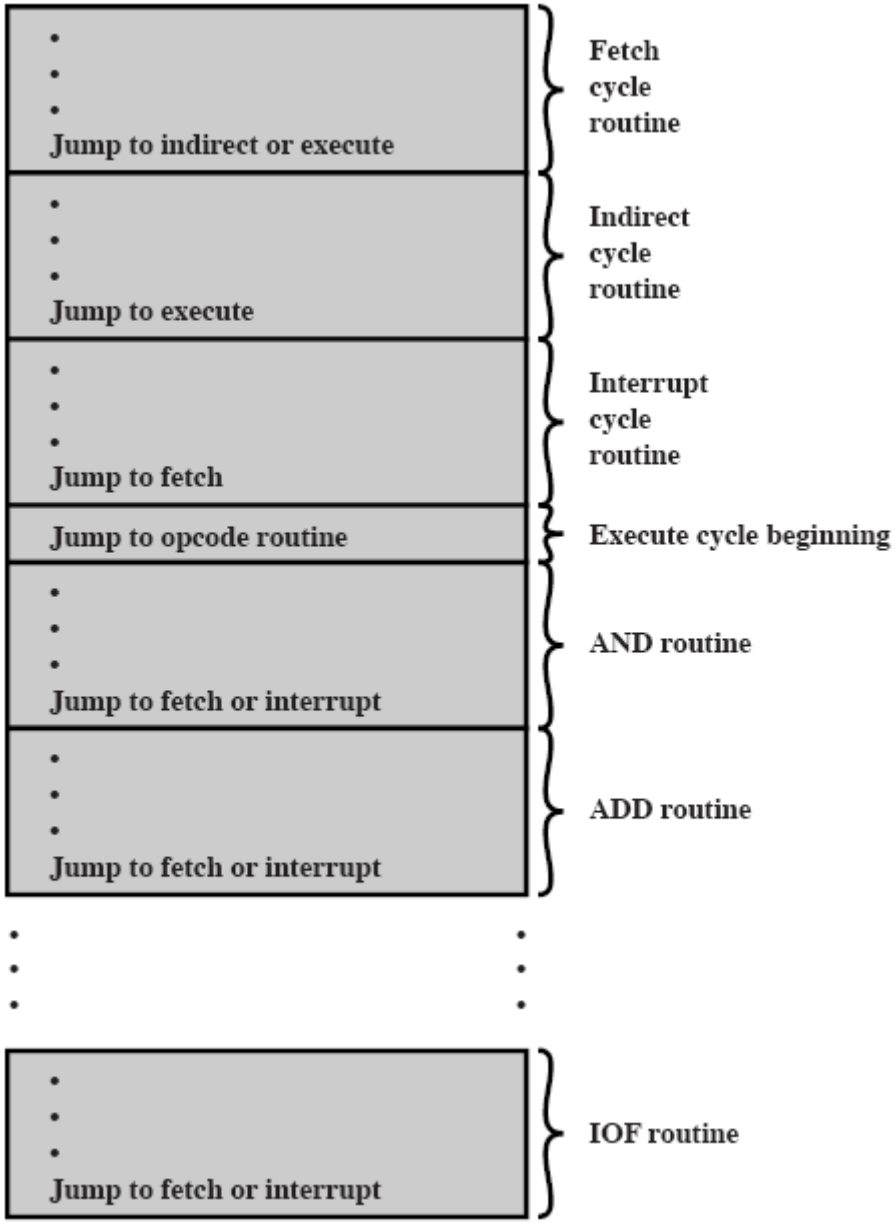
- Idea 1951: Wilkes Microprogrammed Control
- Execution Engine
  - Execution of one machine instruction (or micro-operation) is done by executing a sequence of microinstructions
  - Executes each microinstruction by generating the control signals indicated by the instruction
- Micro-operations stored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

Sta06 Table 16.1 (slide 11)



# Microinstructions

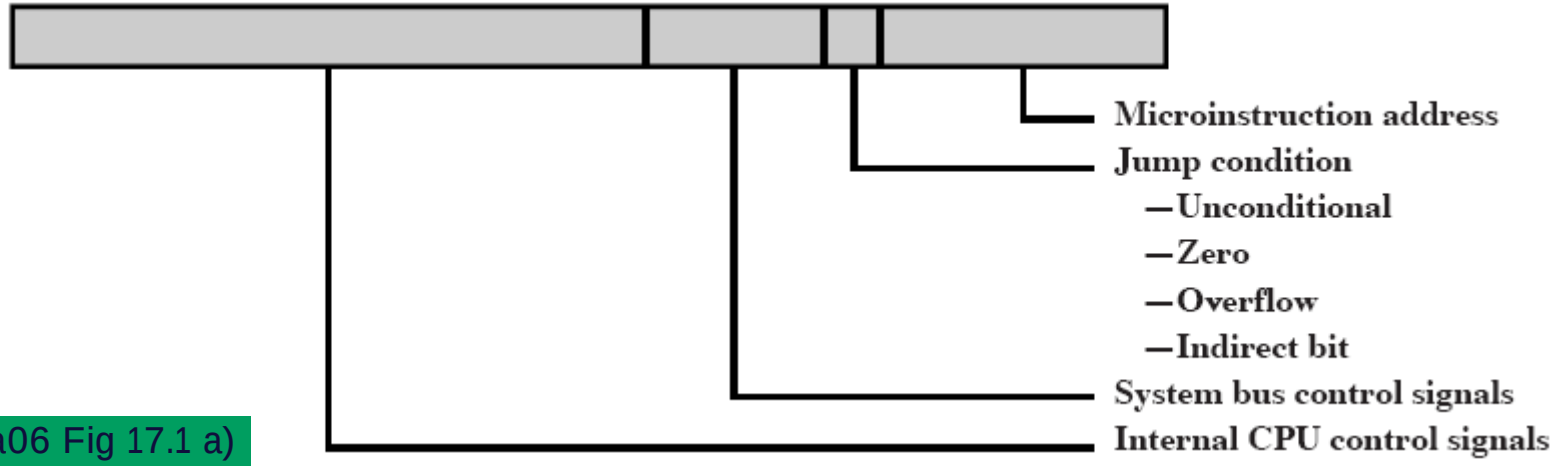
- Each stage in instruction execution cycle is represented by a sequence of microinstructions that are executed during the cycle n that stage
- E.g. In ROM memory
  - **Microprogram or firmware**



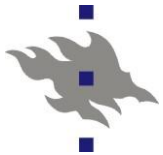

(Sta06 Fig 17.2)

# 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

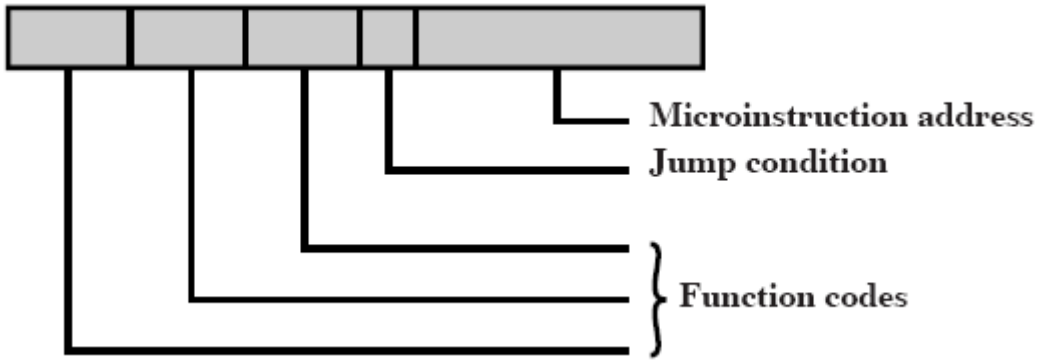



(Sta06 Fig 17.1 a)



# Vertical microinstruction

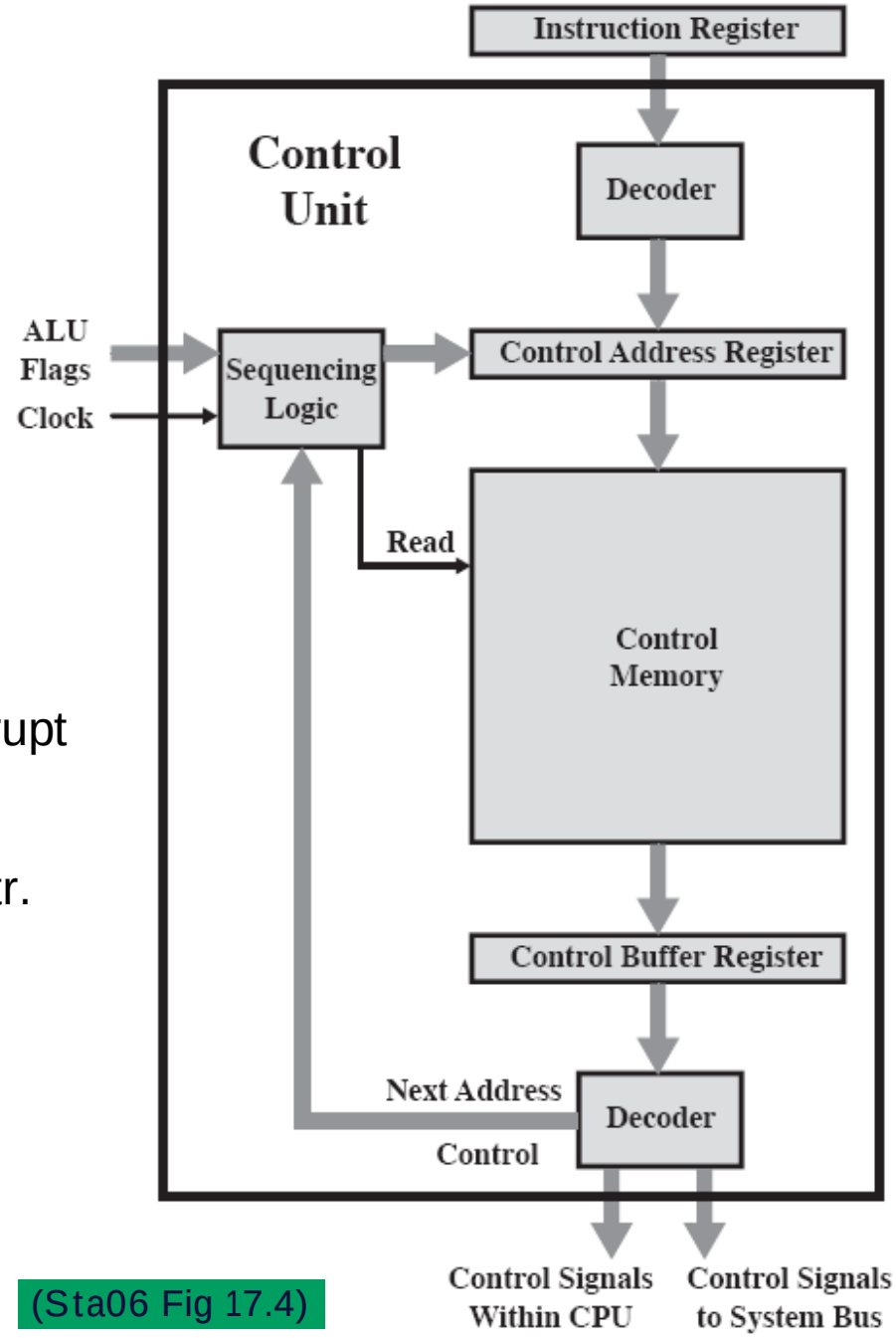
- Control signals coded  to number
- Decode back to control signals during execution
- Shorter instructions, but decoding takes time
- Each microinstruction is conditional branch (as with horizontal instructions)



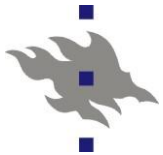

(Sta06 Fig 17.1 b)

# Execution Engine (*Ohjausyksikkö*)

- 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



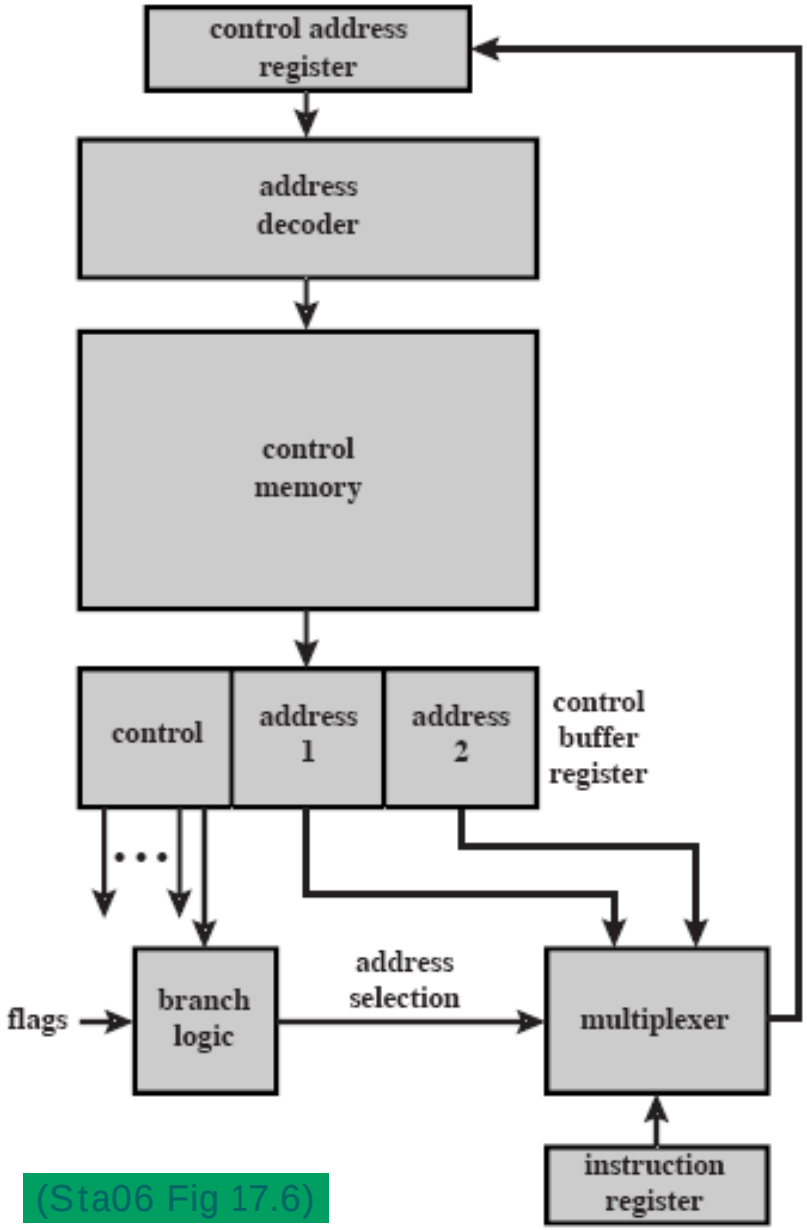

(Sta06 Fig 17.4)



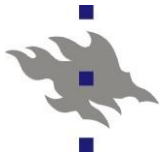
# What microinstruction next?

a) Explicit

- Each instruction has 2 addresses
  - In addition 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!




(Sta06 Fig 17.6)

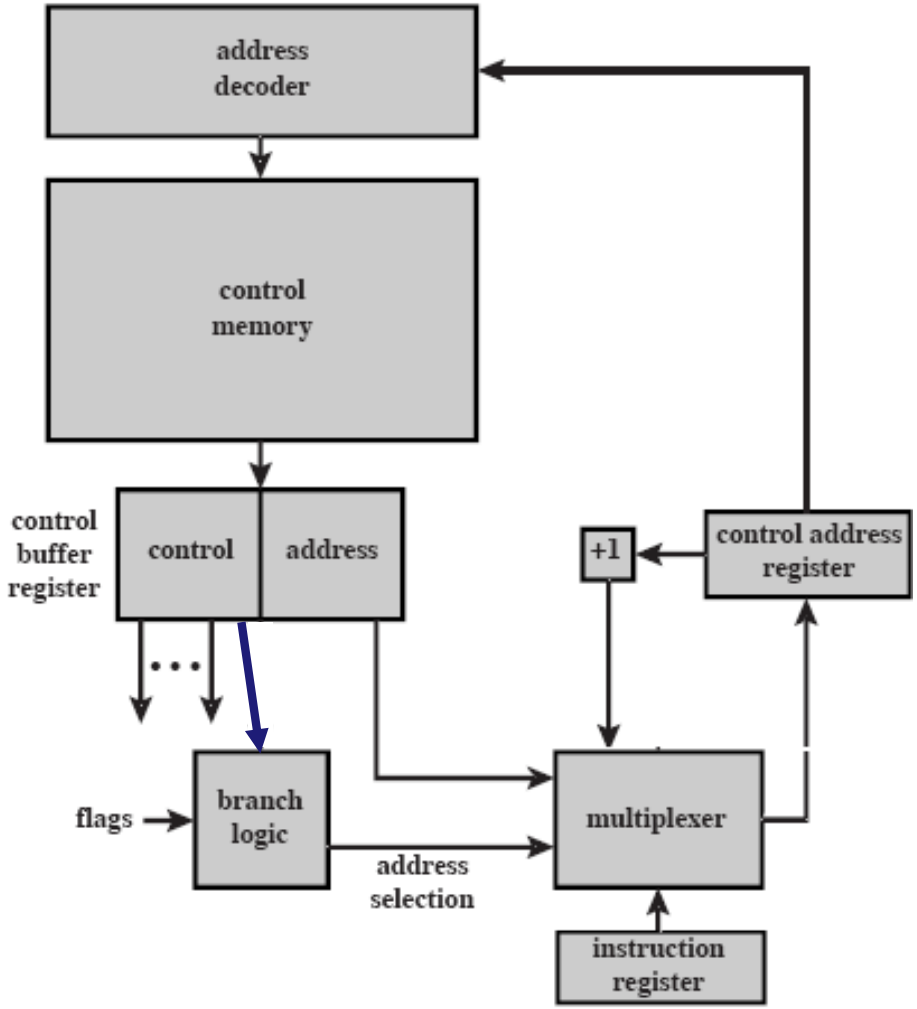


# What microinstruction next?

b) Implicit assumption: next microinstruction from next location in control memory

- instruction has 1 address
  - Still need the condition flags
  - If condition=1,

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

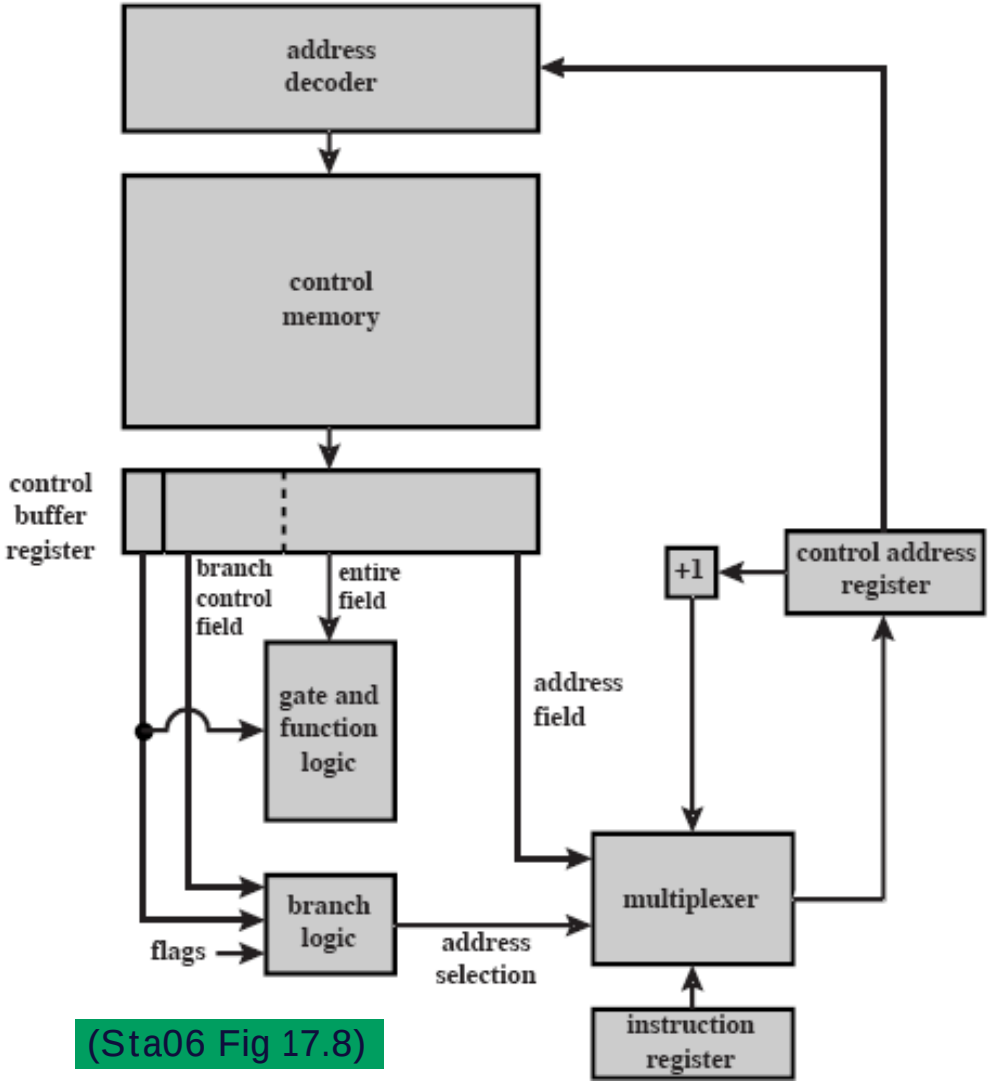



(Sta06 Fig 17.7)

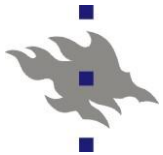
# What microinstruction next?

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?



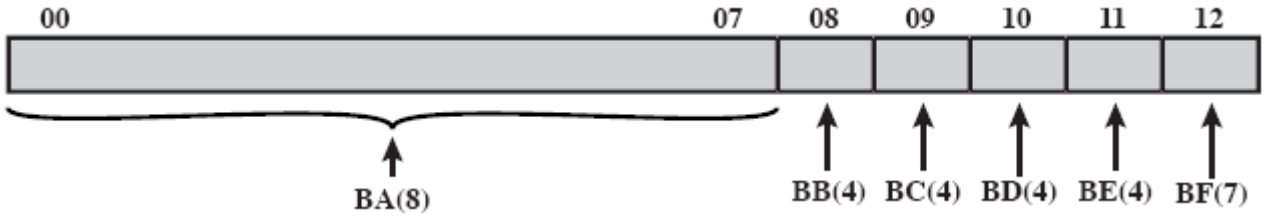

(Sta06 Fig 17.8)



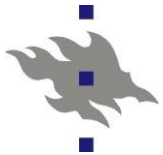
# What microinstruction next?

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-significatn bits based on the current situation (0 or 1)
  - Example: IBM 3033 CAR, 13 bit address
    - Op-code gives 8 bits -> each sequence 32 micro-instr.
    - rest 5 bits based on the certain status bits



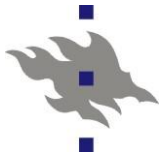

(Sta06 Fig 17.9)



# What microinstruction next?

e) Subroutines and residual control

- Microinstruction can set a special return register 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

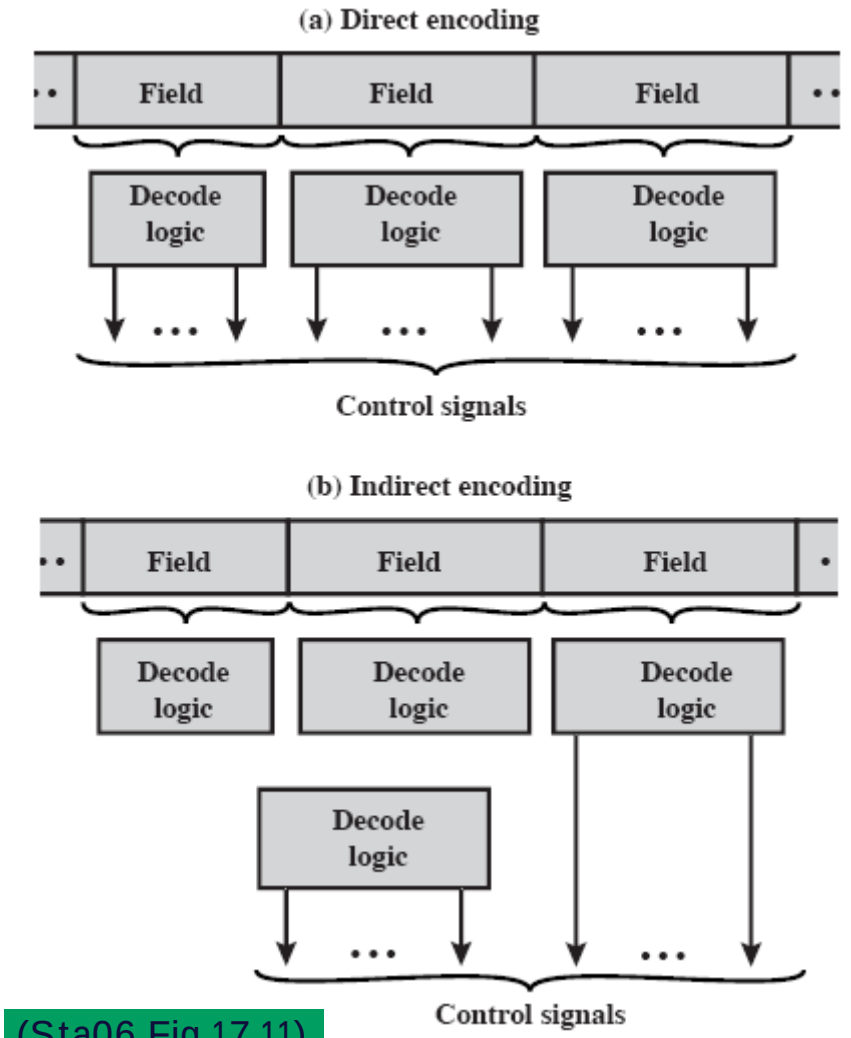


# Microinstruction coding

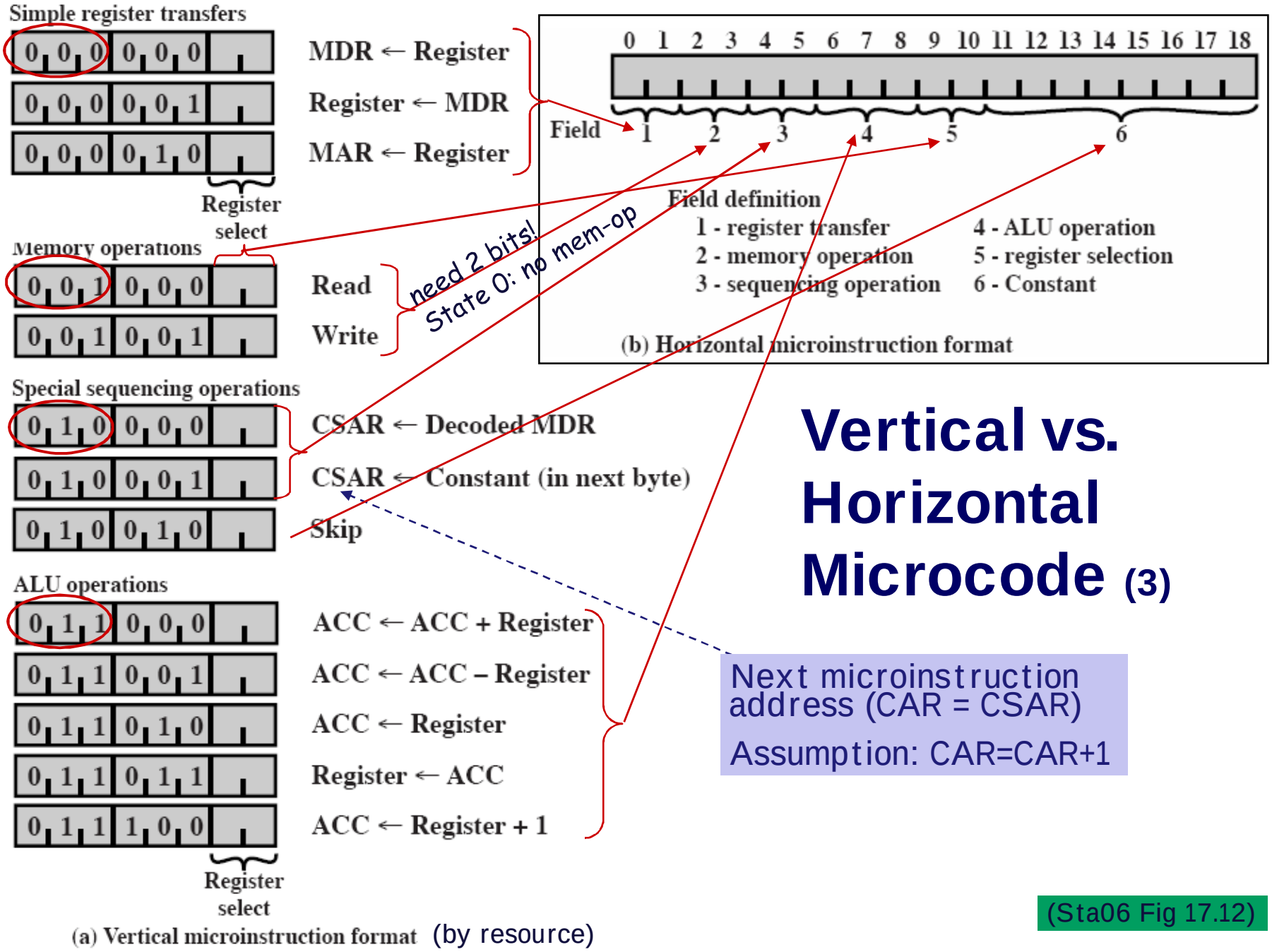
- Horizontal? Vertical?
  - Horizontal: fast interpretation
  - Vertical: less bits, smaller space
- Often a compromize, using mixed model
  - Microinstruction split to fields, each fields is used for certain control signals
  - Excluding signal combinations can be coded in the same field
    - NOT: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

# Microinstruction coding

- Functional encoding (toiminnoittain)
  - Each filed controls one specific action
    - Load from accumulator
    - Load from memory
    - Load from ...
- Resource encoding (resursseittain)
  - Each field controls psecific resourse
    - Load from accumulator
    - Store to accumulator
    - Add to accumulator
    - … accumulator




(Sta06 Fig 17.11)



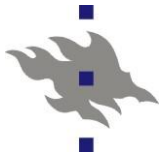

(by resource)

# Vertical vs. Horizontal Microcode (3)

Next microinstruction address (CAR = CSAR)

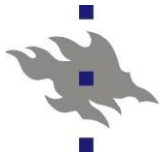
Assumption: CAR=CAR+1

(Sta06 Fig 17.12)



# Why microprogrammed control?

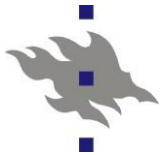
- ..even when its slower than hardwired control
- Design is simple and flexible
  - Modifications (e.g. expansion of instruction set) can be added very late in the design phase
  - Old hardware can be updated by just changing control memory
    - Whole control unit chip in older machines
  - There exist 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



# Review Questions / Kertauskysymyksiä

- Hardwired vs. Microprogrammed control?
- How to determine the addres of microinstruction?
- What is the purpose of control memory?
- Horizontal vs. vertical microinstruction?
- Why not to use microprogrammed control?
- IA-64 control vs. microprogrammed vs. hardwired?

- Langoitettu vs. mikro-ohjelmoitu toteutus?
- Kuinka mikrokäskyn osoite määräytyy?
- Mihin tarvitaan kontrollimuistia?
- Horisontaalinen vs. vertikaalinen mikrokäsky?
- Miksi ei mikro-ohjelmointia?
- IA-64 kontrolli vs. mikro-ohjelmointi vs. langoitettu kontrolli?



# Computer Organization II

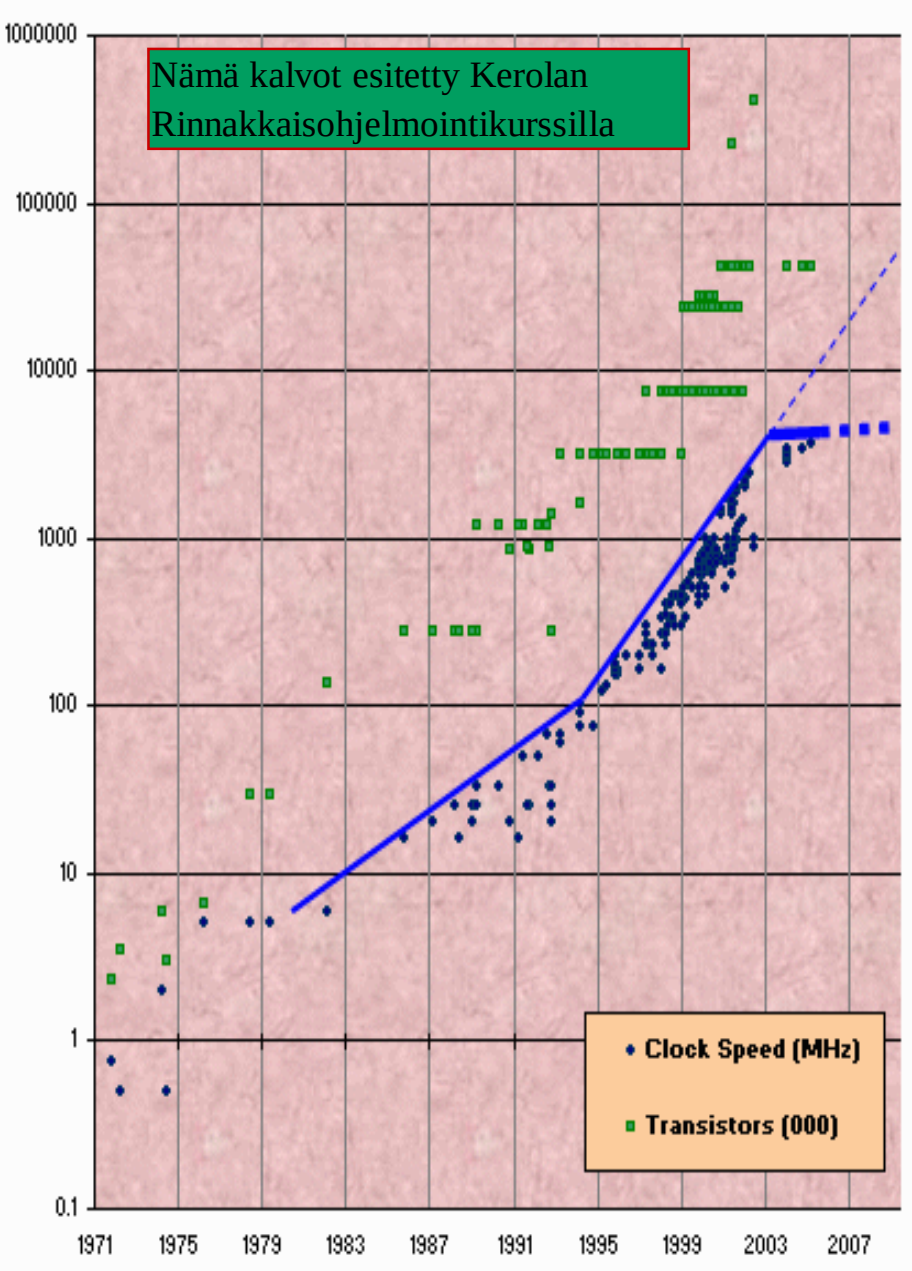
Pääotsikoita olivat

- Digitaalilogiikka
- Väylät, välimuisti, keskusmuisti
- Virtuaalimuistin osoitemuunnos, TLB
- ALU: kokonais- ja liukulukuaritmetiikka
- Käskykannoista: operaatiot ja osoittaminen
- CPU:n rakenne ja liukuhihna
- Hyppyjen ennustus, datariippuvuudet
- RISC & superskalaari CPU, nimiriippuvuudet
- IA-64: Explicit Parallel Instruction Computing
- Langoitettu vs. mikro-ohjelmoitu ohjaus

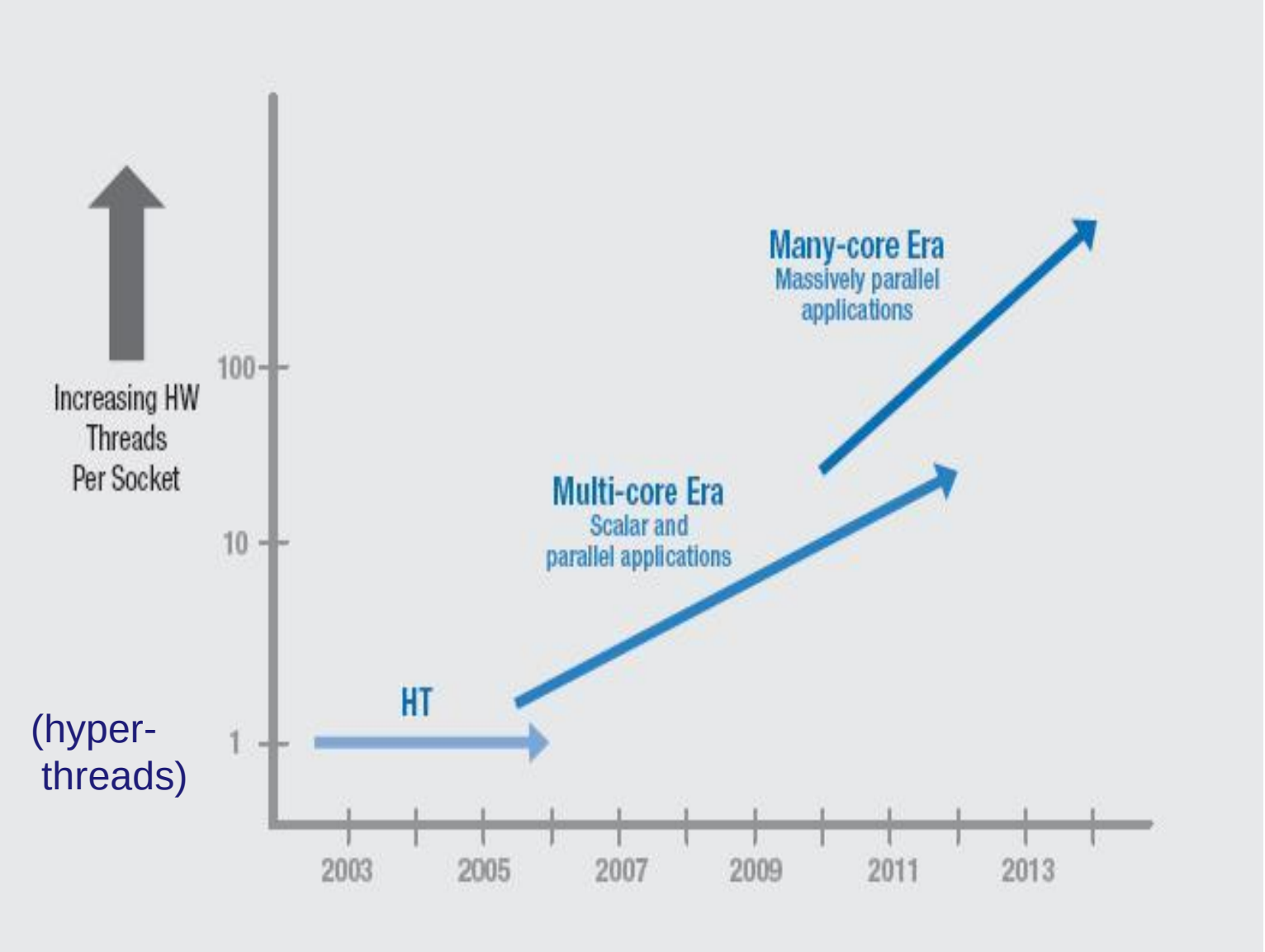
# Problem

- Moore's Law will not give us faster processors (any more)
  - But it gives us now more processors on one chip
    - Multicore CPU
    - Chip-level multiprocessor (CMP)

Herb Sutter, "A Fundamental Turn Toward Concurrency in SW", Dr. Dobb's Journal, 2005. (click)




http://www.ddj.com/web-development/184405990;jsessionid=BW05DMMAOT3ZGQSNDLPCKH0CJUNN2JVN?_requestid=1416784




Borkar, Dubey, Kahn, et al. “Platform 2015.” Intel White Paper, 2005. (click)

http://download.intel.com/technology/computing/archinnov/platform2015/download/Platform_2015.pdf

# -- The End --

STI Cell Power processor element
(a) major units
and
(b) pipeline

http://researchweb.watson.ibm.com/journal/rd/494/kahle2.gif