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?

- 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

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 Signals

Main task: control data transfers
- Inside CPU: REG $\leftrightarrow$ REG, ALU $\leftrightarrow$ REG, ALU-ops
- CPU $\leftrightarrow$ 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

\begin{itemize}
  \item $t_1$: MAR $\leftarrow$ PC
  \item $t_2$: MBR $\leftarrow$ MEM[MAR]
  \item $t_3$: IR $\leftarrow$ (MBR)
\end{itemize}

If implemented without ALU
Instruction cycle (Käskysykli)

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:

1. $t_1$: MAR $\rightarrow$ PC
2. $t_2$: MAR $\rightarrow$ MMU(MAR)
   - Control Bus $\rightarrow$ Reserve
3. $t_3$: Control Bus $\rightarrow$ Read
   - PC $\leftarrow$ PC + 1
4. $t_4$: MBR $\rightarrow$ MEM[MAR]
   - Control Bus $\rightarrow$ Release
5. $t_5$: IR $\leftarrow$ MBR

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?

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

ICC? 00 (fetch) 01 indirect

11 (interrupt)

Setup interrupt

ICC = 00

10 (execute)

Opcode

Read address

Execute instruction

Yes

No

ICC = 11

ICC = 00

No

Interrupt for enabled interrupt?

Yes

Indirect addressing?

No

ICC = 10

ICC = 01

(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

Control signals

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

Discussion?
**Control Signals and Micro-Operations**

<table>
<thead>
<tr>
<th>Micro-operations</th>
<th>Timing</th>
<th>Active Control Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t₁: MAR ← (PC)</td>
<td>C₁₅</td>
<td></td>
</tr>
<tr>
<td>t₂: MBR ← Memory</td>
<td>C₂₆</td>
<td>C₁₅, C₂₆</td>
</tr>
<tr>
<td>t₃: IR ← (MBR)</td>
<td>C₃₇</td>
<td></td>
</tr>
<tr>
<td>Indirect:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t₁: MAR ← (IR(Address))</td>
<td>C₄₈</td>
<td></td>
</tr>
<tr>
<td>t₂: MBR ← Memory</td>
<td>C₅₉, C₆₀</td>
<td></td>
</tr>
<tr>
<td>t₃: IR(Address) ← (MBR(Address))</td>
<td>C₇₁</td>
<td></td>
</tr>
<tr>
<td>Interrupt:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t₁: MBR ← (PC)</td>
<td>C₈₂</td>
<td></td>
</tr>
<tr>
<td>t₂: MAR ← Save-address</td>
<td>C₉₃</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>??</td>
</tr>
<tr>
<td>t₃: Memory ← (MBR)</td>
<td>C₉₄, C₉₅</td>
<td></td>
</tr>
</tbody>
</table>

C₄₈ = Read control signal to system bus.
C₉₄ = Write control signal to system bus.

(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:
1: MAR ← IR.address
2: MBR ← MEM[MAR]
3: Y ← MBR
4: Z ← AC + Y
5: AC ← Z

(Sta10 Fig 15.6)
Hardwired implementation

(Langoitettu ohjaus)

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

Eg. C5 = P*Q*T2 + P*Q*(LDA)*T2 + ...

Fig 15.3, 15.5 and Tbl 15.1

PQ = 00 Fetch Cycle
PQ = 01 Indirect Cycle
PQ = 10 Execute Cycle
PQ = 11 Interrupt Cycle
Hardwired Control Unit

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

<table>
<thead>
<tr>
<th>I1</th>
<th>I2</th>
<th>I3</th>
<th>I4</th>
<th>O1</th>
<th>O2</th>
<th>O3</th>
<th>O4</th>
<th>O5</th>
<th>O6</th>
<th>O7</th>
<th>O8</th>
<th>O9</th>
<th>O10</th>
<th>O11</th>
<th>O12</th>
<th>O13</th>
<th>O14</th>
<th>O15</th>
<th>O16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

C5: opcode = 5 (bits I2, I3, I4) → signal O11 is true (1)

Finite State Diagram

Computer Organization II, Autumn 2010, Teemu Kerola

29.11.2010
### State transitions

<table>
<thead>
<tr>
<th>Next state from current state</th>
<th>Alternatively, prior state &amp; condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 0 -&gt; State 1</td>
<td>S4, S5, S7, S8, S9, S11 -&gt; State 0</td>
</tr>
<tr>
<td>State 1 -&gt; S2, S6, S8, S10</td>
<td></td>
</tr>
<tr>
<td>State 2 -&gt; S5 or …</td>
<td></td>
</tr>
<tr>
<td>State 3 -&gt; S9 or …</td>
<td></td>
</tr>
<tr>
<td>State 4 -&gt; State 0</td>
<td></td>
</tr>
<tr>
<td>State 5 -&gt; State 0</td>
<td></td>
</tr>
<tr>
<td>State 6 -&gt; State 7</td>
<td></td>
</tr>
<tr>
<td>State 7 -&gt; State 0</td>
<td></td>
</tr>
<tr>
<td>State 8 -&gt; State 0</td>
<td></td>
</tr>
<tr>
<td>State 9 -&gt; State 0</td>
<td></td>
</tr>
<tr>
<td>State 10 -&gt; State 11</td>
<td></td>
</tr>
<tr>
<td>State 11 -&gt; State 0</td>
<td></td>
</tr>
<tr>
<td>State 2 &amp; op = SW</td>
<td>State 6 -&gt; State 7</td>
</tr>
<tr>
<td></td>
<td>State 7 -&gt; State 8</td>
</tr>
<tr>
<td>State 3 &amp; op = JMP</td>
<td>State 9 -&gt; State 10</td>
</tr>
<tr>
<td></td>
<td>State 10 -&gt; State 11</td>
</tr>
</tbody>
</table>

### 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*)

- 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 control memory as microinstructions
  - Firmware (*laiteohjelmisto*)
- Each microinstruction has two parts
  - What is done during the coming clock cycle?
    - Microinstruction indicates the control signals
    - 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!
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

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
**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](image1)

---

**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](image2)
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!

Which Microinstruction Next?

b) Implicit
- Assumption: next microinstruction from next location in control memory
  - Must be calculated
- $\mu$Instruction has 1 address
  - Still need condition flags
  - If condition = 1, use the address
- Address part not always used
  - Wasted space
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?

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

![Diagram](Sta10 Fig 16.8)

![Diagram](Sta10 Fig 16.9)
Which 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 or Vertical?
  - Horizontal: fast interpretation
  - Vertical: less bits, smaller space
- Often a compromise, 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 (toiminnolltain)**
  - 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

Vertical vs. Horizontal Microcode (3)

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?