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

If implemented without ALU:

- t1: MAR ← PC
- t2: MBR ← MEM[MAR]
  - PC ← PC + 1
- t3: IR ← (MBR)
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
- t3: Control Bus ← Read  
  - PC ← PC + 1
- t4: MBR ← MEM[MAR]  
  - Control Bus ← Release
- t5: IR ← 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)**

  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

  t1: MBR ← (PC)
  t2: MAR ← Save Address
      PC ← Interrupt_handler_address
  t3: Memory ← (MBR)
Instruction Cycle 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 $\Rightarrow$ CU emits a set of control signals
- Example: processor with single accumulator
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>C₂</td>
<td></td>
</tr>
<tr>
<td>t₂: MBR ← Memory</td>
<td>C₅, Cᵣ</td>
<td>??</td>
</tr>
<tr>
<td></td>
<td>PC ← (PC) + 1</td>
<td></td>
</tr>
<tr>
<td>t₃: IR ← (MBR)</td>
<td>C₄</td>
<td></td>
</tr>
<tr>
<td>Indirect:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t₁: MAR ← (IR(Address))</td>
<td>C₈</td>
<td></td>
</tr>
<tr>
<td>t₂: MBR ← Memory</td>
<td>C₅, Cᵣ</td>
<td></td>
</tr>
<tr>
<td>t₃: IR(Address) ← (MBR(Address))</td>
<td>C₄</td>
<td></td>
</tr>
<tr>
<td>Interrupt:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t₁: MBR ← (PC)</td>
<td>C₁</td>
<td></td>
</tr>
<tr>
<td>t₂: MAR ← Save-address</td>
<td>??</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PC ← Routine-address</td>
<td></td>
</tr>
<tr>
<td>t₃: Memory ← (MBR)</td>
<td>C₁₂, C₇</td>
<td></td>
</tr>
</tbody>
</table>

Cᵣ = Read control signal to system bus.
C₇ = Write control signal to system bus.

(Sta10 Table 15.1)
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

\[ C5 = \text{”read bus to MBR”} \]

- Eg. \( C5 = P \cdot Q \cdot T_2 + \overline{P} \cdot Q \cdot (LDA) \cdot T_2 + \ldots \)

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

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

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

(Sta10 Table 15.3)
State transitions

Next state from current state

State 0 -> State 1
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

__________________ -> State 1
__________________ -> 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 **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

(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
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)

(S10 Fig 16.1 b)
Microinstruction Execution Engine

- Control Address Register, CAR
  - Which microinstruction next?
  - ~ instr. pointer, “μPC”

- Control memory
  - Microinstructions
    - fetch, indirect, execute, interrupt

- Control Buffer Register, CBR
  - Register for executing microinstr.
  - ~ instr. register, “μIR”
  - 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!
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 return register with 'return μaddress'
  - 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
  - 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)

Next microinstruction address (CAR = CSAR)
Assumption: CAR=CAR+1
Why microprogrammed control?

- Even when it's 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 (μcode chip)
  - 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?