Lecture 10: Control Unit

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)
- 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

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

Instruction cycle
- 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: MAR ← PC
2: MAR ← MMU(MAR)
3: Control Bus ← Reserve
4: Control Bus ← Read → MBR
5: PC ← PC + 1
6: MBR ← MEM(MAR)
7: MBR ← MEM(MAR)
8: Control Bus ← Release
9: 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+1 = MEM[MAR]
- 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

Control Unit

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

Control Signals and Micro-Operations

- Micro-operations © CU emits a set of control signals
- Example: processor with single accumulator

Control Unit as State-Machine

- ICC: Instruction Cycle Code register’s state

Control signals

- Flags
- Sequencing
- Execution

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
Hardwired implementation (Langoitettu ohjaus)

Hardwired Control Unit
- Opcode decoder (4-to-16)
  - 4-bit instruction code as input to CU
  - Only one signal active at any given stage
- Hardwired implementation of CU
  - CU's inputs and outputs fixed
  - Functionality described using Boolean logic
  - CU implemented by one logical circuit
  - Example: C5 = P*Q*T2 + P*Q*(LDA)*T2 + ...

Finite State Diagram

State transitions

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 sign,
    - 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

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

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)
Lecture 10: Control Unit

10.2.2012

Comp. Org II, Spring 2012
Microinstruction Coding

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

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
- 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?