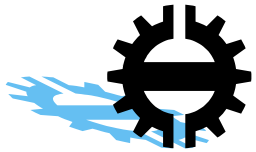
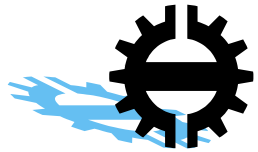


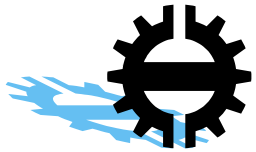
Processor Design
–
Control and Interrupts

Professor Jari Nurmi
Institute of Digital and Computer Systems
Tampere University of Technology, Finland
email jari.nurmi@tut.fi



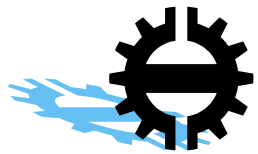
Processor Control Design

- Instruction sequencing control
- Instruction decoding
- Pipeline control
- Handling interrupts and exceptions



Instruction Sequencing Control

- PC points to the instruction to be fetched
  - The central component in sequencing
- PC gets its value from
  - Incrementer (linear progress)
  - Instruction immediate (absolute branch)
  - ALU output (computed branch)
  - Register file or dedicated register (indirect branch/return)
  - Interrupt handling logic (interrupt or exception)
- The sequencing control is basically a logic block
  - To control a multiplexer at the PC input
  - To control the loading of PC (e.g. stalls or memory wait states!)
- Control logic input from
  - Instruction decode (linear or absolute/computed/indirect branch?)
  - Pipeline control (stall?)
  - Interrupt handler (interrupt/exception?)



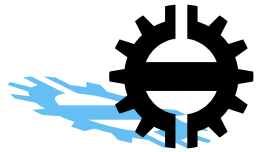
Branching on Arithmetic Conditions

- An ALU with two m-bit operands produces more than just an m-bit result
- The carry from the left bit and the true/false value of 2's complement overflow are useful
- There are 3 common ways of using outcome of compare (subtract) for a branch condition

  1) Do the compare in the branch instruction

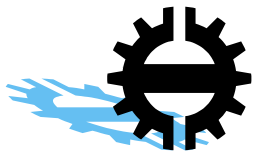
  2) Set special condition code bits and test them in the branch

  3) Set a general register to a comparison outcome and branch on this logical value



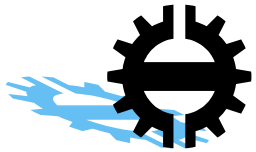
Drawbacks of Condition Codes

- Condition codes are extra processor state; set and overwritten by many instructions
- Setting and use of CCs also introduces hazards in a pipelined design
- CCs are a scarce resource; they must be used before being set again
  - The PowerPC has 8 sets of CC bits
- CCs are processor state that must be saved and restored during exception handling



Drawbacks of Comparison in Branch and Set General Register

- Branch instruction length: it must specify 2 operands to be compared, branch target, and branch condition (possibly place for link)
- Amount of work before branch decision: it must use the ALU and test its output—this means more branch delay slots in pipeline
- Setting a general register to a particular outcome of a compare, say $\leq$ unsigned, uses a register of 32 or more bits for a true/false value



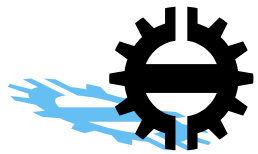
Use of Condition Codes: MC68000

- The HLL statement:

  if (A > B) then C = D

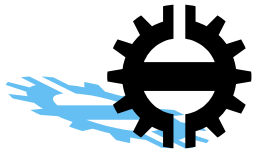
  translates to the MC68000 code:

For 2's comp. A and B	For unsigned A and B
`MOVE.W A, D0`	`MOVE.W A, D0`
`CMP.W B, D0`	`CMP.W B, D0`
`BLE Over`	`BLS Over`
`MOVE.W D, C`	`MOVE.W D, C`
`Over: . . .`	`Over: . . .`



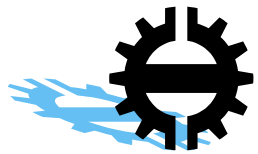
Standard Condition Codes: NZVC

- Assume compare does the subtraction s = x - y
- N: negative result, $s_{m-1} = 1$
- Z: zero result, s = 0
- V: 2's complement overflow, $x_{m-1}y_{m-1}\bar{s}_{m-1} + \bar{x}_{m-1}\bar{y}_{m-1}s_{m-1}$
- C: carry from leftmost bit position, $s_m = 1$
- Information in N, Z, V, and C determines several signed & unsigned relations of x and y



Correspondence of Conditions and NZVC Bits

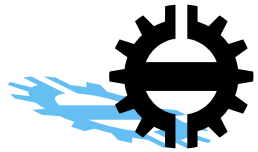
Condition	Unsigned Integers	Signed Integers
carry out	C	C
overflow	C	V
negative	n.a.	N
>	$\overline{C} \cdot \overline{Z}$	$(N \cdot V + \overline{N} \cdot \overline{V}) \cdot \overline{Z}$
≥	C	$N \cdot V + N \cdot V$
=	Z	Z
≠	$\overline{Z}$	$\overline{Z}$
≤	C+Z	$(N \cdot \overline{V} + \overline{N} \cdot V) + Z$
<	C	$N \cdot \overline{V} + \overline{N} \cdot V$



Branches That Do Not Use Condition Codes

- SRC (in Heuring-Jordan book) compares a single number to zero
- The simple comparison can be completed in pipeline stage 2
- The MIPS R2000 compares 2 numbers using a branch of the form: `bgtu R1, R2, Lbl`
- Different branch instructions are needed for each signed or unsigned condition
- The MIPS R2000 also allows setting a general register to 1 or 0 on a compare outcome

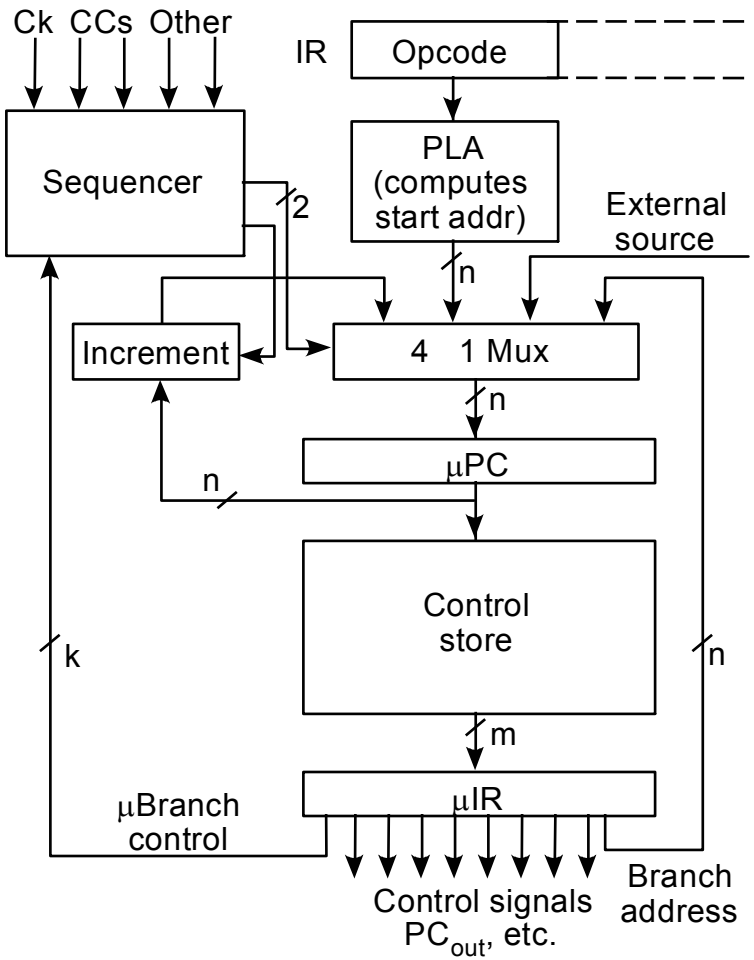
```
sgtu R3, R1, R2
```



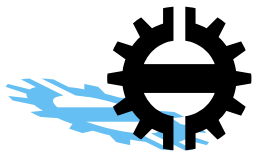
Instruction Decoding

- Extraction of all individual control signals from the instruction word
- The control signals within an instruction cycle
  - partially static
  - partially depend on the clock phase
  - may depend on conditions (e.g. branch or conditional execution)
- Different implementation styles, e.g.
  - Microcoded control
  - Hardwired logic

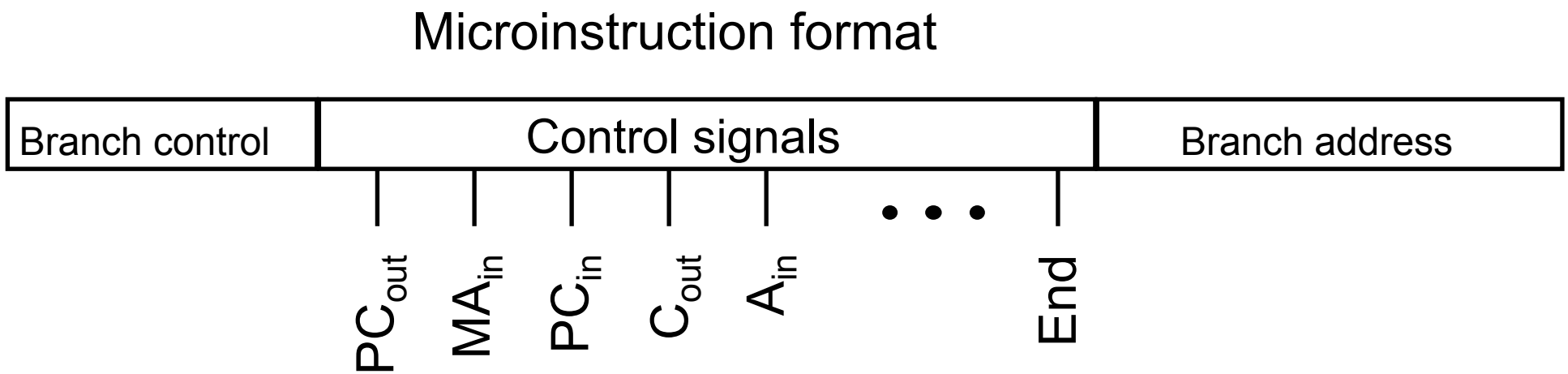
Block Diagram of Microcoded Control Unit



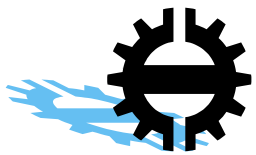
- Microinstruction has branch control, branch address, and control signal fields
- Microprogram counter can be set from several sources to do the required sequencing



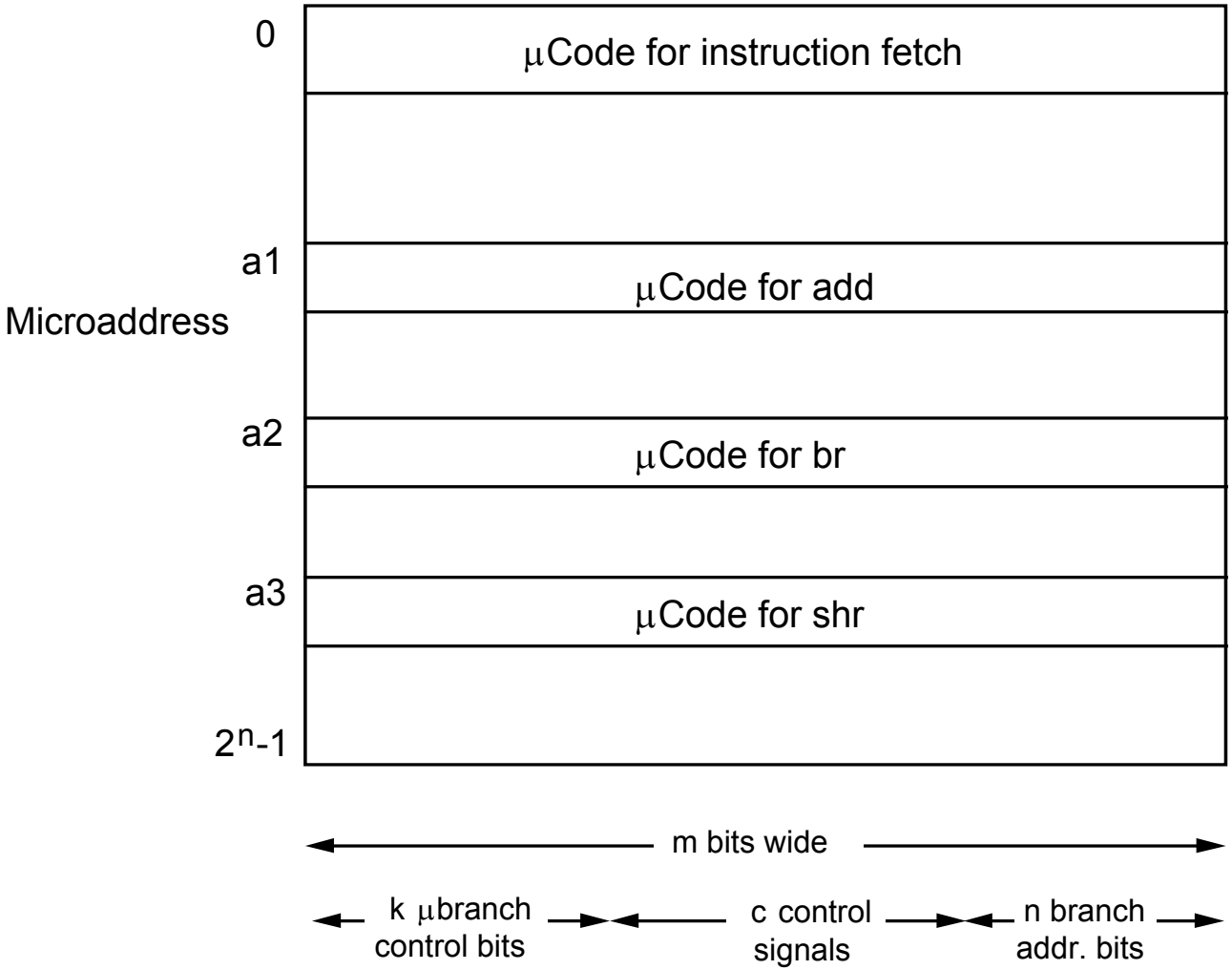
Contents of a Microinstruction



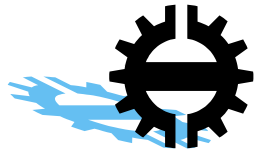
- Main component is list of 1/0 control signal values
- There is a branch address in the control store
- There are branch control bits to determine when to use the branch address and when to use μPC + 1



The Control Store

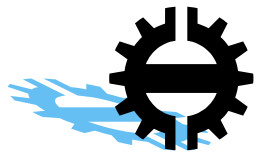


- Common instruction fetch sequence
- Separate sequences for each (macro) instruction
- Wide words



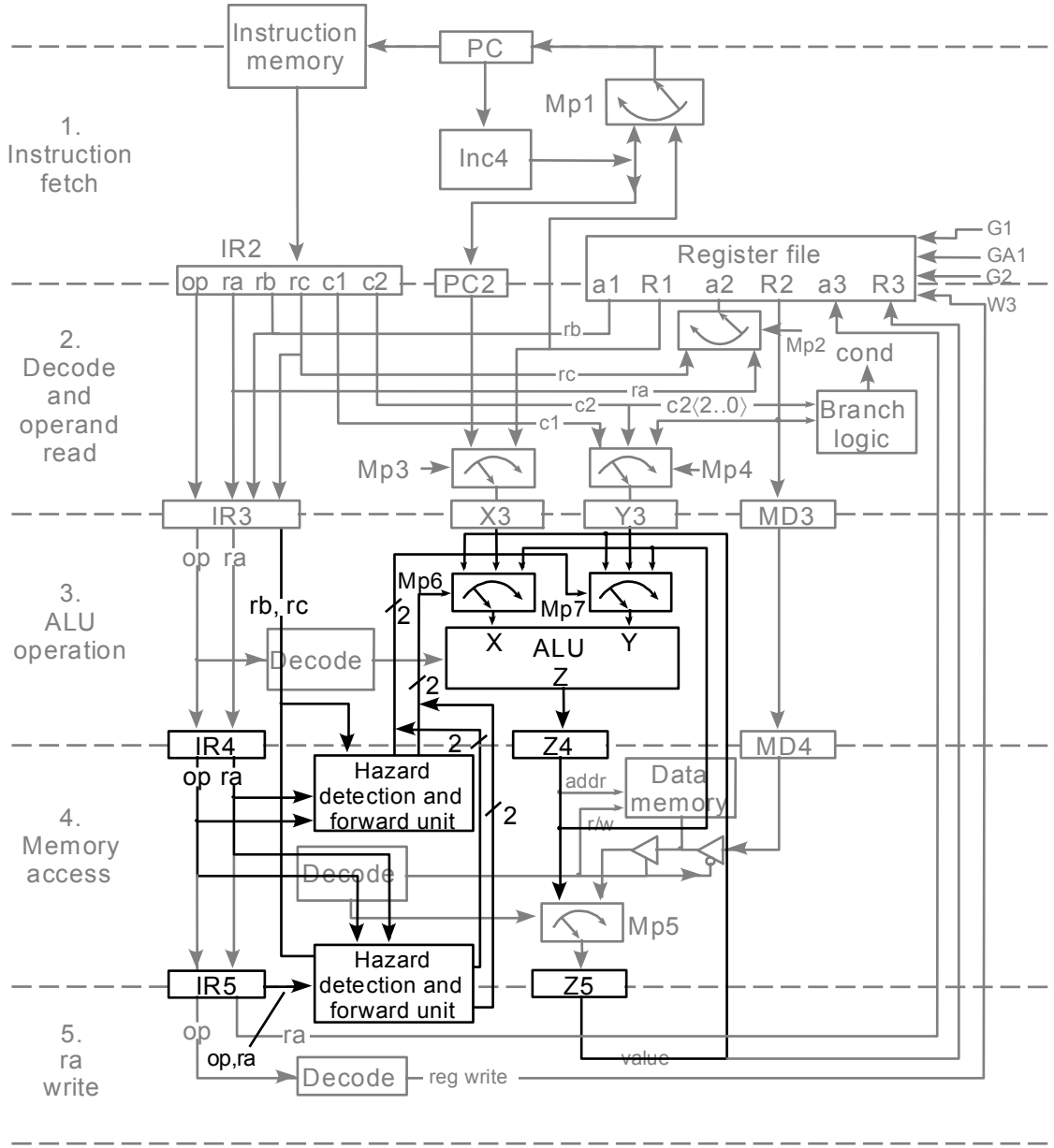
Hardwired Instruction Decoding

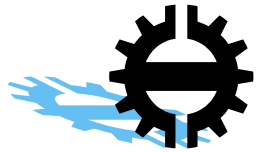
- Microcoded control implies multiple clock cycles (or multiple phases) to execute one instruction
- For single-cycle instructions, the hardwired decoder is a simpler choice
  - Logic mapping instruction codes (and operand fields) to control signals
  - Phase-dependent control signals handled by ANDing with clock
- The logic can be implemented as
  - A PLA structure
  - Random gates (synthesized!)



Pipeline Control consists of

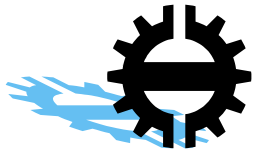
- Decoded instruction pipeline registers
- Logic to detect
  - Need for forwarding
  - Need for stalls
- Logic to implement
  - Forward multiplexer control
  - Stall insertion control
- Multiplexers for forwarding





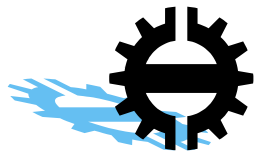
Handling Interrupts and Exceptions

- Some terminology
  - Interrupt is an external event requesting service from the processor
  - Exception is an internal error situation calling for immediate repair
    - E.g. overflow, memory protection violation, illegal opcode,…
    - Can occur in fetch, decode, execute, data memory access
- In both cases, the response is to execute a service routine, either immediately or at the earliest convenience
  - Exceptions cannot wait, the next instructions after the faulty one may not change the processor state
  - External interrupts have a latency requirement typically in the range of microseconds
- The interrupt can be
  - “plain”, meaning that there is a single interrupt service routine for a physical interrupt line coming in (maybe polling what caused the interrupt)
  - Vectored, meaning that the interrupt carries along an index that is used for selecting the correct service routine
- Interrupts are independent of the program flow (except software interrupts, traps) and can take place at any time, also simultaneously



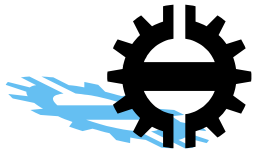
Interrupt Handler

- The interrupt handler takes care of arbitration and nesting
  - Higher priority interrupts can interrupt a service routine
  - Lower/same priority interrupts will remain pending
  - Interrupts may be disabled for short periods to guarantee correct transition between the interrupted program flow and interrupt service
  - Interrupts (individual or different priorities) can be masked
  - The most crucial ones are non-maskable
- The Interrupt Handler hardware is basically a complex state machine
  - Taking care of who gets service and when
    - Priorities, masking, arbitration of simultaneous events
  - Operating at every cycle of the processor
  - Checking for internal (or external) requests for service
  - Providing the correct interrupt service routine address to PC



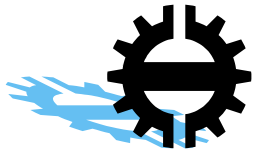
Processor Support for Interrupts

- Of course, the Interrupt Handler hardware is needed for
  - signaling the valid interrupt (for the sequencing control)
  - providing the service address
- HW for saving return address and status register implicitly when an interrupt is signalled
- ... and restoring them when returning from the routine (”return from interrupt” instruction)
- All the rest can be handled by software
  - saving working registers needed by the service routine
  - the actual routine (whatever it does)
  - restoring the working register contents



Summary

- This lecture has dealt with processor control design
- The main parts of control are
  - Instruction sequencing control (branch techniques)
  - Instruction decoding
  - Pipeline control
  - Handling interrupts and exceptions
- Especially, interrupts make things more complicated and must be watched after every cycle by specific interrupt hardware



End of Control and Interrupts

next we will look at Memory Subsystem