# Computer Architecture TDTS10

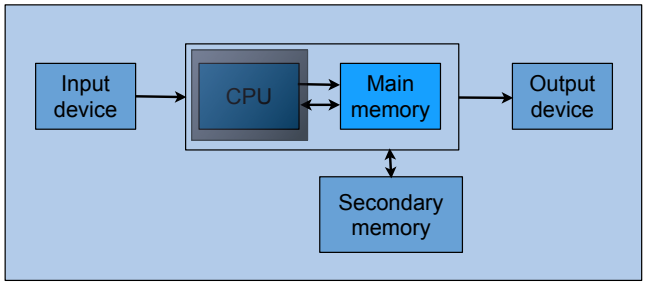
Erik Larsson
Department of Computer Science
Linköping University
Sweden

## Outline

- Control unit
- Input/Output Devices and System Buses
- Programmed I/O, Interrupt-driven I/O, and Direct Memory Access
- RISC and CISC

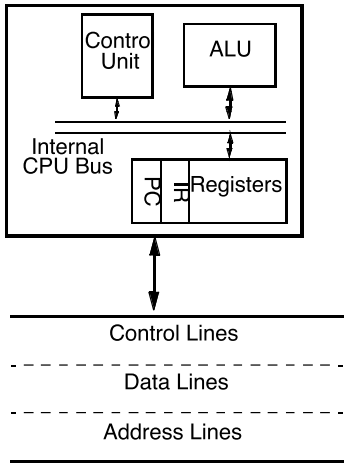


## CPU

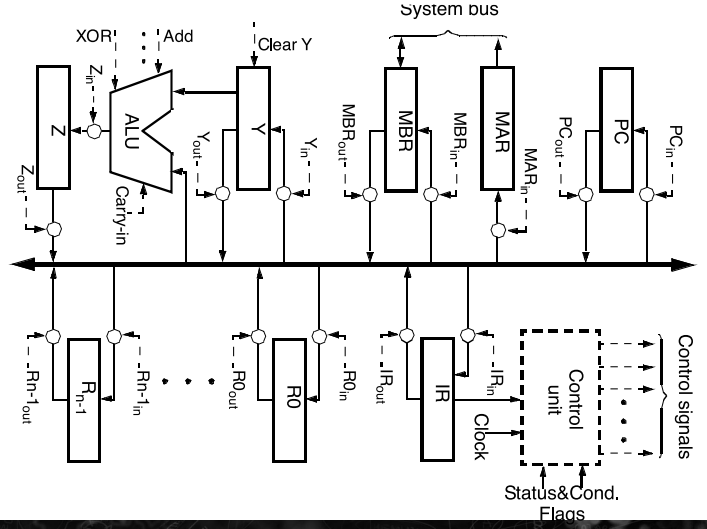





## Internal Structure of the CPU



# Internal Structure of the CPU






# Internal Structure of the CPU

- The CPU executes an instruction as a sequence of control steps. In each control step one or several microoperations are executed.
- Execution of a microoperation, one or several control signals have to be issued;
  - a) signals for transferring content of register R0 to R1: R0out, R1in
  - b) signals for adding content of Y to that of R0 (result in Z): R0out, Add, Zin
  - c) signals for reading a memory location; address in R3: R3out, MARin, Read
- One clock pulse triggers the activities corresponding to one control step -> for each clock pulse the control unit generates the control signals corresponding to the microoperations to be executed in the respective control step.



# Microoperations and Control Signals

- instruction:
  - ADD R1,R3 R1 <- R1 + R3
- control steps and control signals:

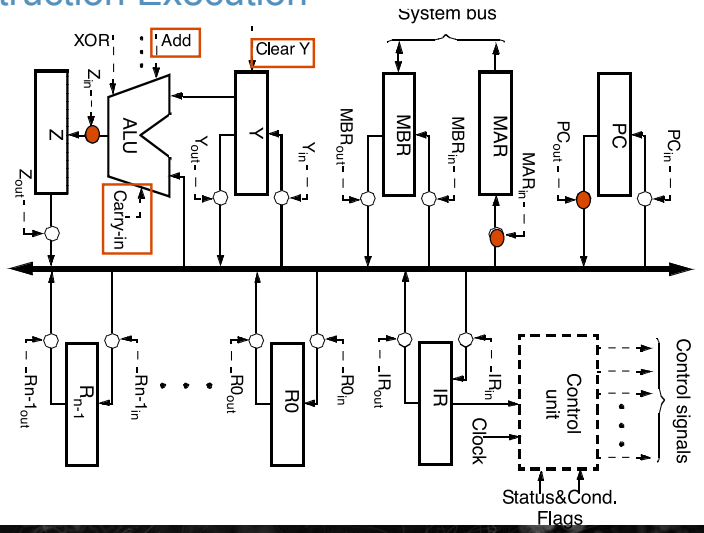
fetch ins. PC <- PC+1

1. PCout, $MAR_{in}$, Read, Clear Y, Carry-in, Add, $Z_{in}$
2. $Z_{out}$, $PC_{in}$
3. $MBR_{out}$, $IR_{in}$
4. $R1_{out}$, $Y_{in}$
5. $R3_{out}$, Add, $Z_{in}$
6. $Z_{out}$, $R1_{in}$, End



PCout, MARin, Read, Clear Y, Carry-in, Add, Zin

# Instruction Execution



## Microoperations and Control Signals

- instruction:
  - ADD R1,R3 R1 <- R1 + R3
- control steps and control signals:

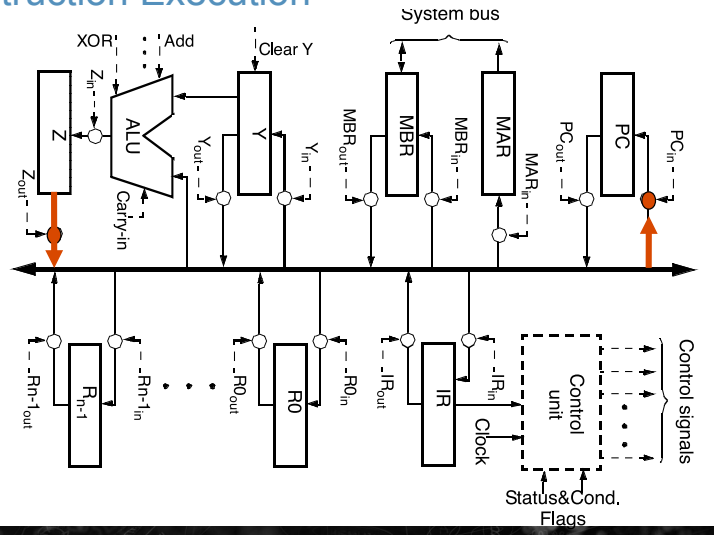
fetch ins. PC← PC+1

1. PCout, $MAR_{in}$, Read, Clear Y, Carry-in, Add, $Z_{in}$
2. $Z_{out}$, $PC_{in}$
3. $MBR_{out}$, $IR_{in}$
4. $R1_{out}$, $Y_{in}$
5. $R3_{out}$, Add, $Z_{in}$
6. $Z_{out}$, $R1_{in}$, End



Zout, PCin

## Instruction Execution






## Microoperations and Control Signals

- instruction:
  - ADD R1,R3 R1 <- R1 + R3
- control steps and control signals:

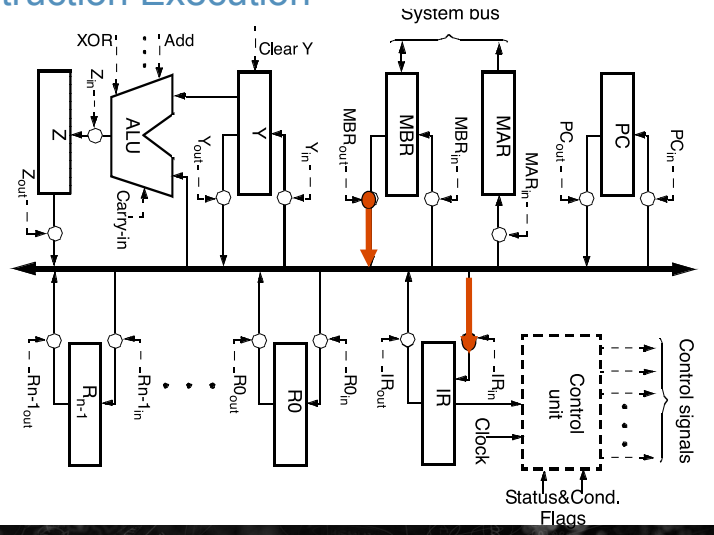
fetch ins. PC← PC+1

1. PCout, $MAR_{in}$, Read, Clear Y, Carry-in, Add, $Z_{in}$
2. $Z_{out}$, $PC_{in}$
3. $MBR_{out}$, $IR_{in}$
4. $R1_{out}$, $Y_{in}$
5. $R3_{out}$, Add, $Z_{in}$
6. $Z_{out}$, $R1_{in}$, End



MBRout, IRin

## Instruction Execution



## Microoperations and Control Signals

- instruction:
  - ADD R1,R3 R1 <- R1 + R3
- control steps and control signals:

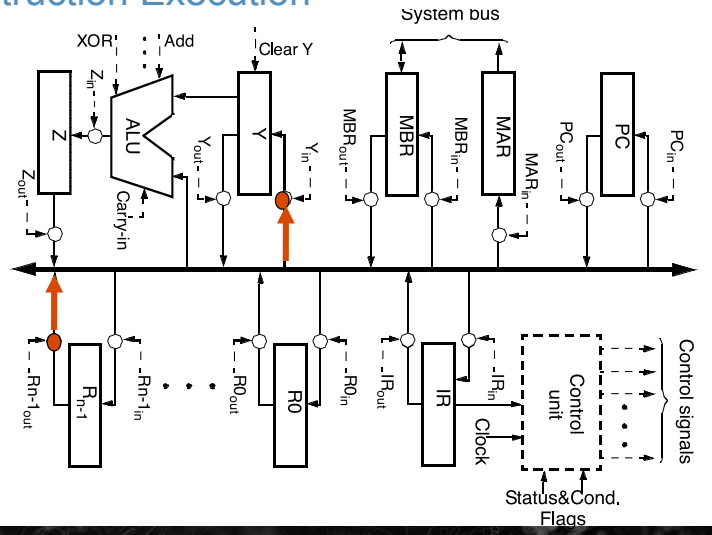
fetch ins. PC← PC+1

1. PCout, $MAR_{in}$, Read, Clear Y, Carry-in, Add, $Z_{in}$
2. $Z_{out}$, $PC_{in}$
3. $MBR_{out}$, $IR_{in}$
4. $R1_{out}$, $Y_{in}$
5. $R3_{out}$, Add, $Z_{in}$
6. $Z_{out}$, $R1_{in}$, End



R1out, Yin

## Instruction Execution






## Microoperations and Control Signals

- instruction:
  - ADD R1,R3 R1 <- R1 + R3
- control steps and control signals:

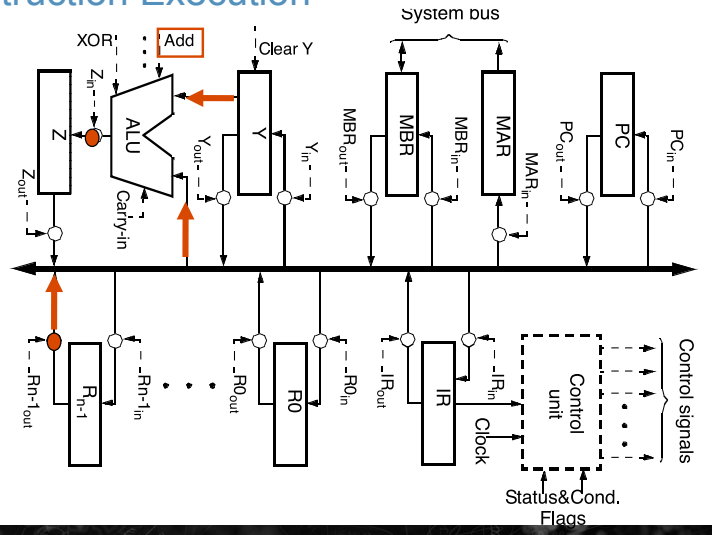
fetch ins. PC← PC+1

1. PCout, $MAR_{in}$, Read, Clear Y, Carry-in, Add, $Z_{in}$
2. $Z_{out}$, $PC_{in}$
3. $MBR_{out}$, $IR_{in}$
4. $R1_{out}$, $Y_{in}$
5. $R3_{out}$, Add, $Z_{in}$
6. $Z_{out}$, $R1_{in}$, End



R3out, Add, Zin

## Instruction Execution



## Microoperations and Control Signals

- instruction:
  - ADD R1,R3 R1 <- R1 + R3
- control steps and control signals:

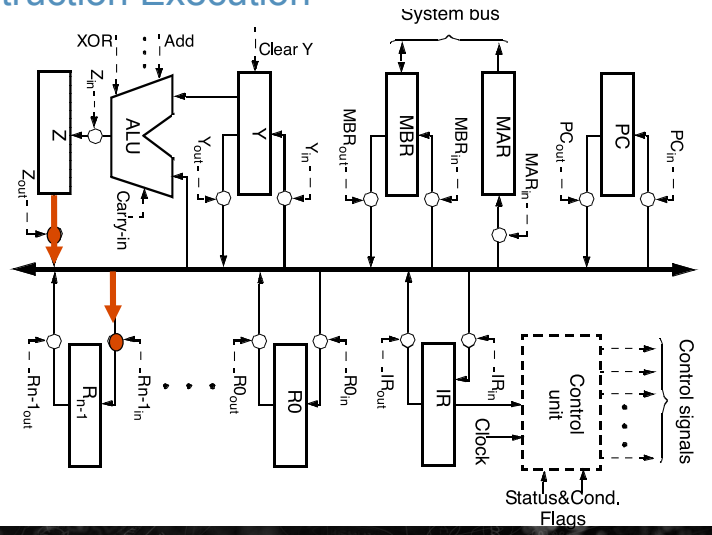
fetch ins. PC <- PC+1

1. PCout, $MAR_{in}$, Read, Clear Y, Carry-in, Add, $Z_{in}$
2. $Z_{out}$, $PC_{in}$
3. $MBR_{out}$, $IR_{in}$
4. $R1_{out}$, $Y_{in}$
5. $R3_{out}$, Add, $Z_{in}$
6. $Z_{out}$, $R1_{in}$, End



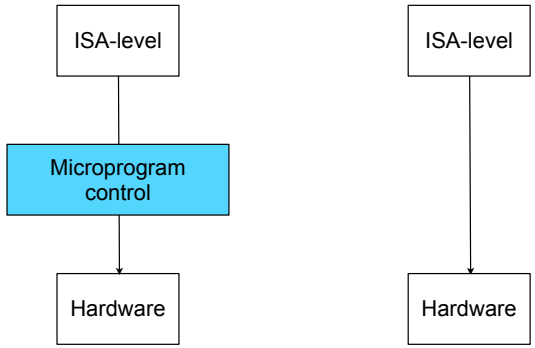


## Instruction Execution



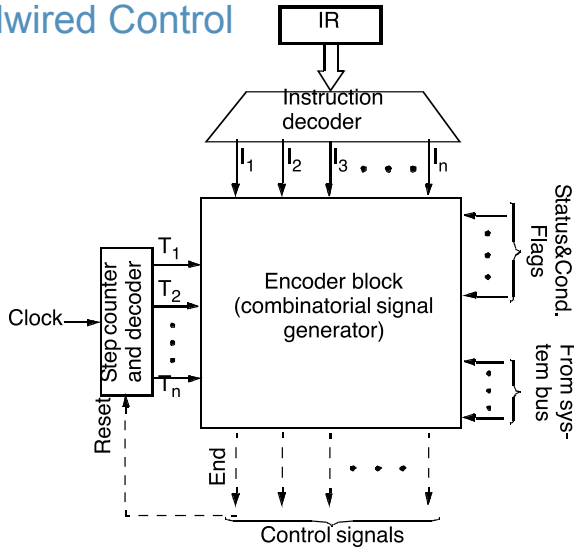



## Implementation of instruction set architecture (ISA)

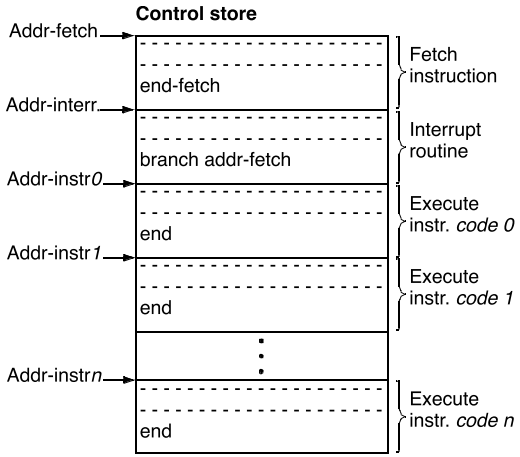





## Hardwired Control



## Control Store Organization






## Summary

- The control unit coordinates the CPU by issuing in each clock cycle the appropriate control signals.
- Control signals activates the microoperations
- Control units can be hardwired or microprogrammed.
- A hardwired control unit is a combinatorial circuit
- A microprogrammed control unit is implemented like another CPU inside the CPU.
- Hardwired controllers are faster than microprogrammed.
- Microprogrammed controllers can implement advanced instructions

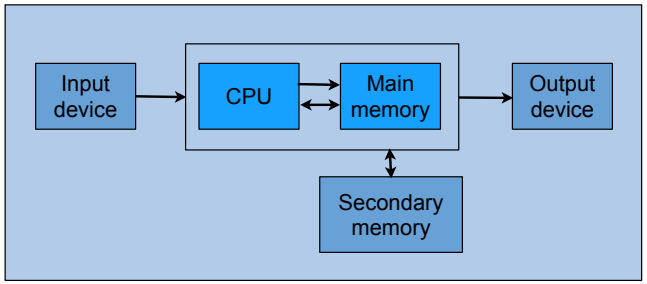


## Outline

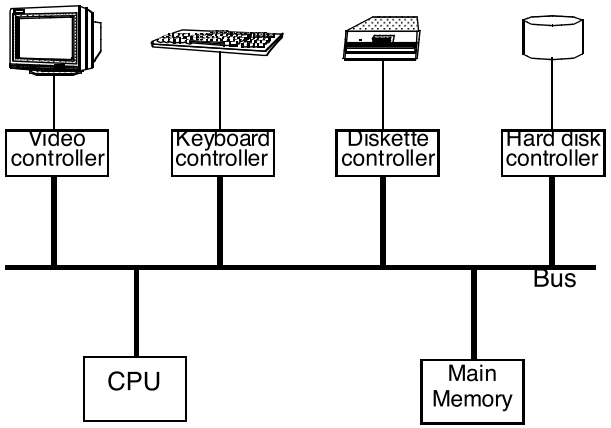
- Control unit
- Input/Output Devices and System Buses
  - Bus organization
    - Arbitration, timing
  - CPU interface
  - I/O interface
- Programmed I/O, Interrupt-driven I/O, and Direct Memory Access
- RISC and CISC



## Computer system

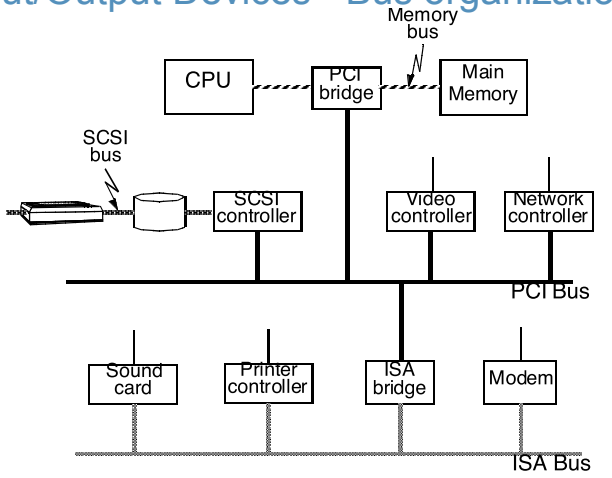


## Input/Output Devices - Bus organization

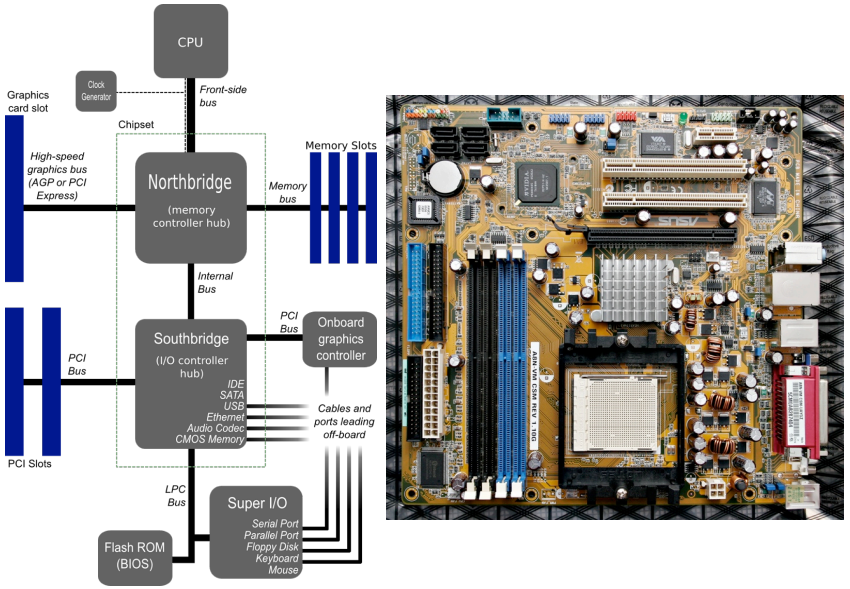





## Input/Output Devices - Bus organization








## System Buses

- A bus - 50-100 separate lines/wires
- Classified into three functional groups:
  - Data lines: moving data between system components.
  - Address lines: are used to designate the source or destination of data.
  - Control lines: are used to control bus access, synchronize operations, and to propagate commands throughout the system.
- In order to avoid large buses -> multiplexed bus.
- Multiplexed bus:
  - Advantage: Bus width can be reduced
  - Disadvantage: The system becomes slower

## Bus Arbitration

- Devices connected to a bus can be of two kinds:
  - Master: is active and can initiate a bus transfer.
  - Slave: is passive and waits for requests.
- Some devices can act both as master and as slave, depending on the circumstances:
  - CPU is typically a master.
  - A coprocessor, however, can initiate a transfer of a parameter from the CPU -> CPU acts like a slave.
  - An I/O device usually acts like a slave in interaction with the CPU.
  - Several devices can perform direct access to the memory, in which case they access the bus like a master.
  - The memory acts only like a slave.



## Bus Arbitration

- Since only one unit at a time can transmit over the bus, arbitration is needed.
- Arbitration mechanisms:
  - Centralized arbitration: there is a single device, the bus arbiter, that determines who goes next.
  - Decentralized (distributed) arbitration: no arbiter is needed.
- Examples:
  - PCI and ISA buses use a centralized arbitration scheme.
  - SCSI buses use a decentralized scheme.



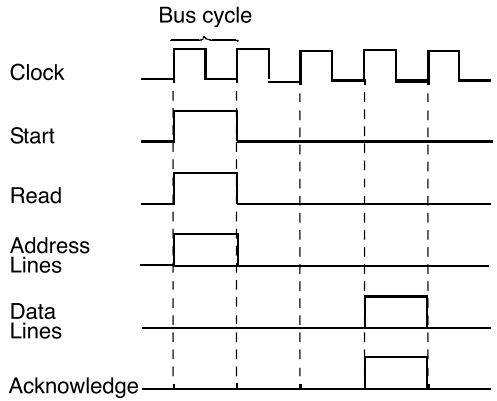
## Bus Timing

- Timing refers to the way in which events are coordinated on the bus:
  - Synchronous timing: the occurrence of events on the bus is determined by a clock.
  - Asynchronous timing: the occurrence of one event on a bus follows and depends on the occurrence of a previous event.
- Examples:
  - PCI and ISA buses use synchronous timing.
  - SCSI buses use asynchronous timing.



## Synchronous Timing

Adopt to slowest device
Easy to design

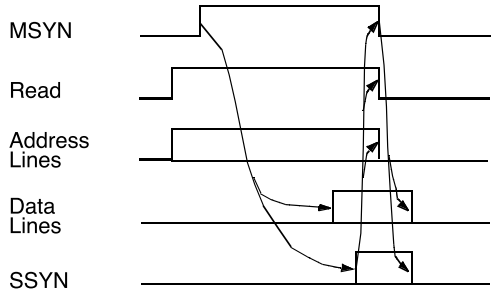


## Synchronous Timing

- The bus includes a clock line; all devices on the bus can read the clock line.
- All events on the bus start at the beginning of a clock cycle.
- A bus sequence for a synchronous memory read.
- The CPU (master) issues a start signal to mark the presence of address and control information on the bus: the read signal is issued on the respective control line, and the memory address is placed on the address lines.
- After a delay of two bus cycles, the memory (slave) places the data on the data lines and issues an acknowledge signal on the respective control line.
- Adopt to slowest device. Easy to design



## Asynchronous Timing






## Asynchronous Timing

- There is no clock line on the bus.
- Each event is caused by a prior event, not by the clock pulse. The master will wait exactly as much as is needed for the slave to finish.
- If a master has to wait long for a certain slow slave, this does not influence how much it will have to wait for.
- A bus sequence for an asynchronous memory read.
  - 1. CPU (master) asserted the address lines and issue read signal
  - 2 wait until lines are stable and then issue MSYN signal (Master SYNchronization).
  - 3. memory (slave) sees the MSYN, performs the work and asserts the SSYN (Slave SYNchronization) signal.
  - 4 When the master has noticed the SSYN, it knows that data is on the lines and latches



## Input/Output Devices - Bus organization

- CPU and memory connected by local bus
- Industry Standard Architecture (ISA) bus
- Peripheral Component Interconnect (PCI) bus
- Peripheral Component Interconnect Express (PCI Express)
- Accelerated Graphics Port (AGP)
- Small Computer System Interface (SCSI) bus
- Universal Serial Bus (USB)
- IEEE 1394 (Firewire (Apple), i.LINK (Sony) och DV (Panasonic))
- Thunderbolt

## Input/Output Devices - Bus organization

- A bus is a common electrical pathway between multiple devices. In addition to such "system buses", there are buses also inside the CPU (internal buses).
- System buses differ in the number and organization of lines, arbitration, timing, and specific bus operations.
- Different buses are connected through adequate bridges (bridges also perform buffering of information);
- Advantages of architectures with multiple buses:
  - avoids bus conflicts;
  - insulates CPU-to-memory traffic from I/O traffic;
  - allows the system to support a variety of I/O devices tailored for different bus standards.
- In order to connect a device to a bus, the device controller must fit to the respective bus features.



## Input/Output Devices - Bus organization

- Bus conflict -> bus arbiter decides on access.
- I/O devices are given preference over the CPU; usually devices cannot be stopped -> forcing them to wait would result in loss data.
- When no I/O is in progress, the CPU has all bus cycles for itself to reference memory.
- When some I/O device is also running and requests the bus, it gets it -> cycle stealing slows down the computer.

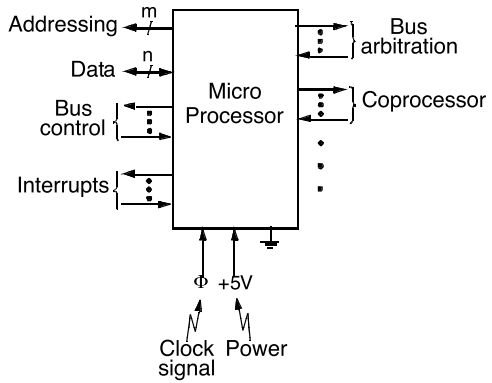


## Outline

- Control unit
- Input/Output Devices and System Buses
  - Bus organization
    - Arbitration, timing
  - CPU interface
  - I/O interface
- Programmed I/O, Interrupt-driven I/O, and Direct Memory Access
- RISC and CISC



## External Interface of the CPU Chip



## External Interface of the CPU Chip

- The CPU pins can be divided into: address pins, data pins, and control pins.
  - address pins: the address is output to the system bus on these pins, for read/write operations. With m address pins, 2^m locations can be addresses.
  - data pins: data bits are output/received to/from the system bus on these pins.
  - with n data pins an n-bit word can be read written in a single operation.
- control pins:
- bus control: the CPU uses these pins to control the rest of the system and tell it what it wants to do; control signals are propagated over the system bus.
- interrupt pins: on these pins the CPU gets signals from I/O modules; they usually indicate that an I/O operation has been completed;
- bus arbitration: are needed to regulate traffic on the system bus, to prevent devices from trying to use it at the same time;
- coprocessor: facilitate communication with coprocessors, such as floating point chips, graphic chips, etc.

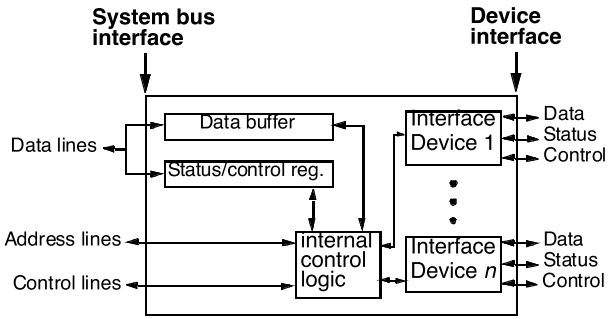


## Outline

- Control unit
- Input/Output Devices and System Buses
  - Bus organization
    - Arbitration, timing
  - CPU interface
  - I/O interface
    - access, I/O processing
- Programmed I/O, Interrupt-driven I/O, and Direct Memory Access
- RISC and CISC



## I/O Modules

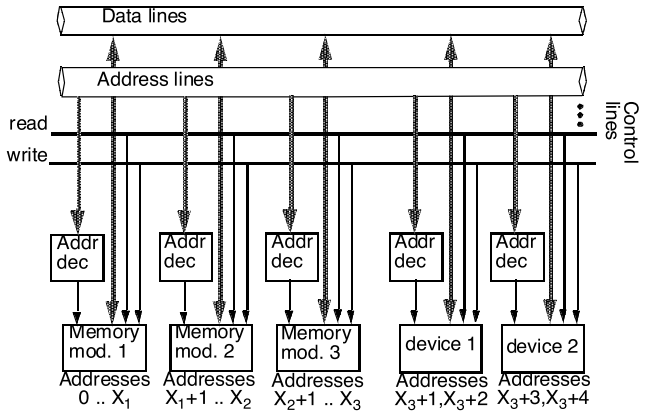





## I/O Modules

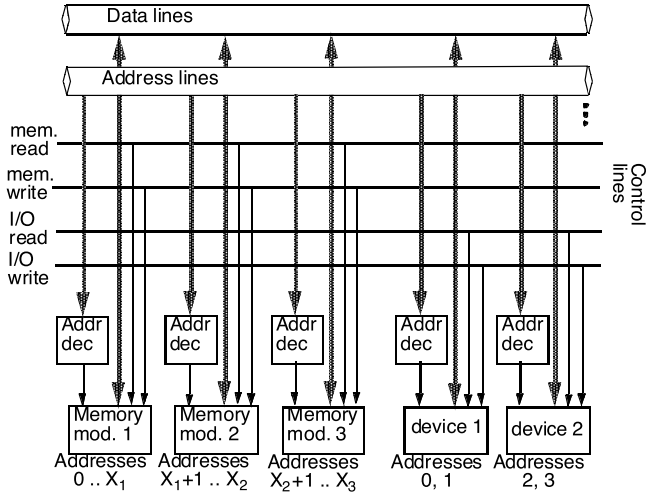
- An I/O module has an interface to the device and to the system bus
- Major functions of an I/O module:
  - control and timing of the operations;
  - bus communication;
  - device communication;
  - data buffering;
  - error detection.
- A possible sequence data transfer between a device and the CPU:
  - CPU interrogates the status of I/O module (device).
  - I/O module returns device status.
  - If the device is OK and ready, the CPU requests the transfer of data by means of a command to the I/O module.
  - The I/O module issues commands to the device and obtains data.

# Memory-mapped I/O






# Isolated I/O



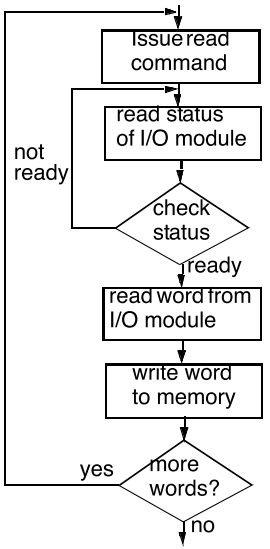



# I/O Processing

- Techniques for I/O:
  - Programmed I/O
  - Interrupt-driven I/O
  - Direct memory access

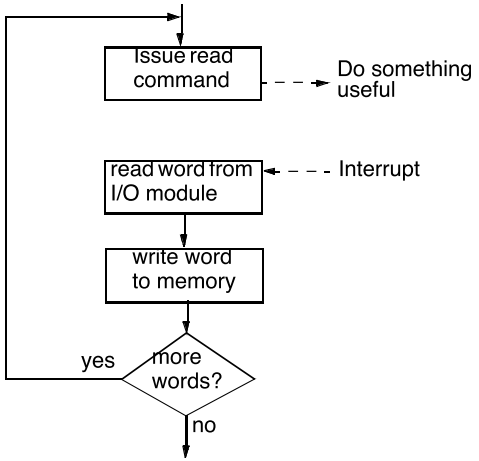


# Programmed I/O



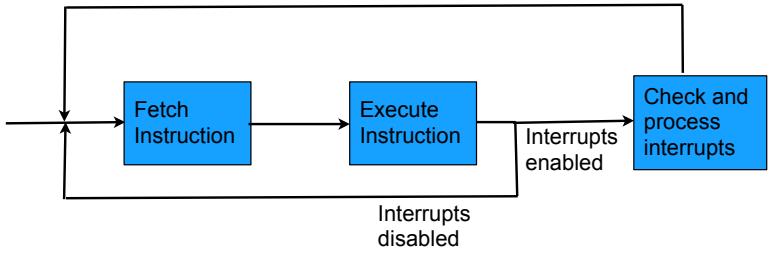
## Interrupt-driven I/O

OS is involved



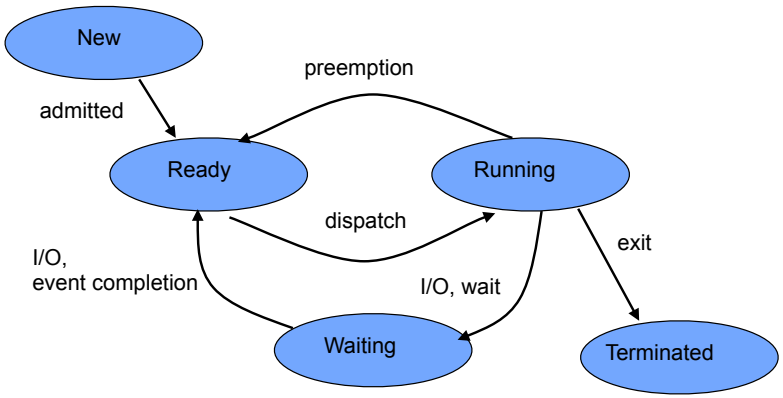



## Interrupts



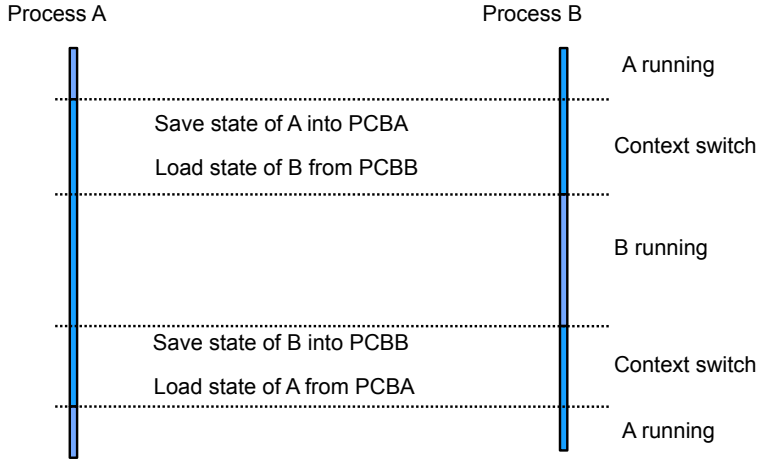



## Process States

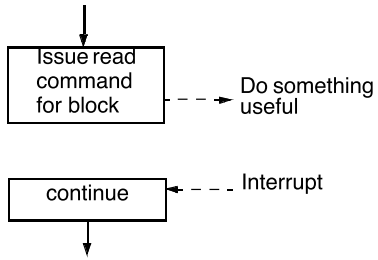





## Context Switch

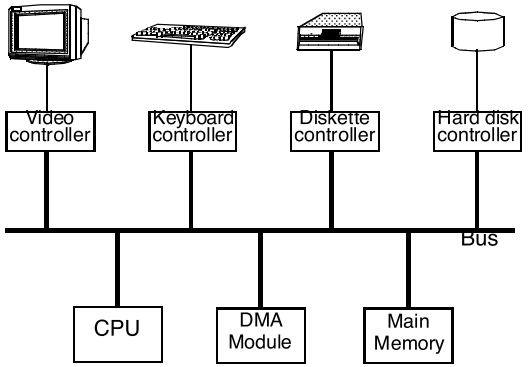


## Direct Memory Access (DMA)






## Direct Memory Access (DMA)






## Summary

- CPU, memory and I/O devices are connected by system buses.
- The CPU chip is connected through address, data, and control pins.
- A bus consists of data, address, and control lines
- Bus arbitration can be centralized or decentralized.
- Bus coordination can be synchronous or asynchronous.
- I/O modules interface an I/O device to the system bus.
- I/O device can be memory-mapped or isolated I/O.
- Techniques for I/O: programmed I/O, interrupt-driven I/O, and direct memory access.

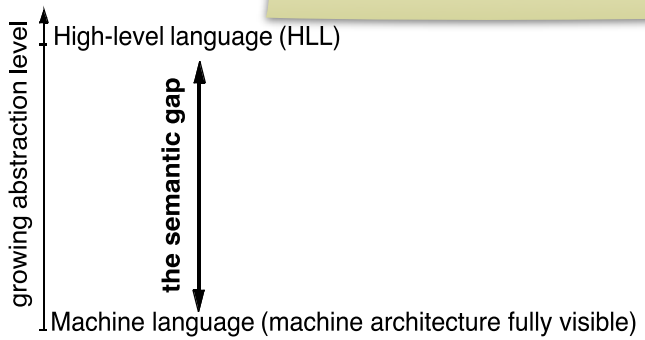


## Outline

- Control unit
- Input/Output Devices and System Buses
- Programmed I/O, Interrupt-driven I/O, and Direct Memory Access
- RISC and CISC
  - The problem and motivation
  - Register file
  - Instruction set
  - Pipeline

## Semantic gap

In order to improve the efficiency of software development, new and powerful programming languages have been developed (Ada, C, C++) Other languages like Java also exists. The more advanced languages provide: high level of abstraction, conciseness, power.






## Semantic gap

- Problem: How should HLL be compiled and executed on an architecture?
- Two directions:
  - Complex instruction set computers (CISC) - complex architecture with a large number of instructions and addressing modes to be close to HLL
  - Reduced instruction set computers (RISC) - simpler architecture and few instructions and addressing modes so that execution is faster



## Motivation

The source code contains this amount of instructions

For each type, this is the amount of machine instructions.

For each type, this is the amount of memory references.

| | Occurrence | | Machine-instruction weighted | | Memory-reference weighted | |
|---|---|---|---|---|---|---|
| | Pascal | C | Pascal | C | Pascal | C |
| Assign | 45% | 38% | 13% | 13% | 14% | 15% |
| Loop | 5% | 3% | 42% | 32% | 33% | 26% |
| Call | 15% | 12% | 31% | 33% | 44% | 45% |
| If | 29% | 43% | 11% | 21% | 7% | 13% |
| Other | 6% | 1% | 3% | 1% | 2% | 1% |

- Conclusions:
  - There are many assign constructions (X=5, Y=X+Z, ...) in a HLL, but each such instruction results in few machine instructions, often with few memory references.
  - On the other hand, there are only few subroutine/procedure/etc (call/return) but each such translates into a high number of machine instructions, with many memory references.



## Conclusions

- Common with simple (ALU and move) instructions
- Common with simple addressing modes
- Large frequency of operand accesses; on average each instruction references 1.9 operands
- Most of the referenced operands are scalars (so they can be stored in a register) and are local variables or parameters
- Optimizing the procedure CALL/RETURN mechanism promises large benefits in speed

## Outline

- Control unit
- Input/Output Devices and System Buses
- Programmed I/O, Interrupt-driven I/O, and Direct Memory Access
- RISC and CISC
  - The problem and motivation
  - Register file
  - Instruction set
  - Pipeline

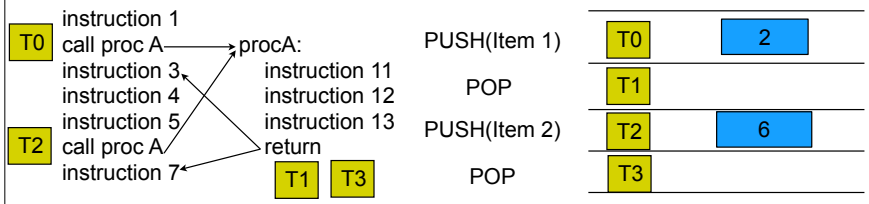


## Program execution analysis

- Procedure Calls
  - Even if only 15% of the HLL instructions are CALL or RETURN, they are executed most of the time, because of their complexity.
  - A CALL or RETURN is compiled into a relatively long sequence of machine instructions with a lot of memory references.
- Some statistics concerning procedure calls:
  - Only 1.25% of called procedures have more than six parameters.
  - Only 6.7% of called procedures have more than six local variables.
  - Chains of nested procedure calls are usually short and only very seldom longer than 6.



## Alternative 1: Stack

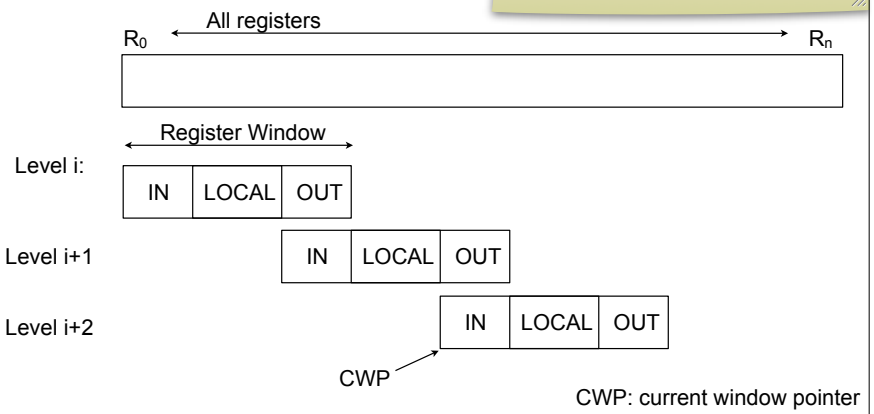



- PUSH/POP: accesses the memory where the stack is



## Alternative 2: Registers

High number of registers makes it possible to store return address and parameters in registers. Instead of making use of time consuming memory accesses (cache or main memory) to store/load parameters, registers offers a fast alternative




CWP: current window pointer

## High number of registers

- Variables and intermediate results can be stored in registers and do not require repeated loads and stores from/to memory.
- All local variables of procedures and the passed parameters can be stored in registers



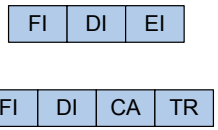
## Outline

- Control unit
- Input/Output Devices and System Buses
- Programmed I/O, Interrupt-driven I/O, and Direct Memory Access
- RISC and CISC
  - The problem and motivation
  - Register file
  - Instruction set
  - Pipeline



## RISC architecture

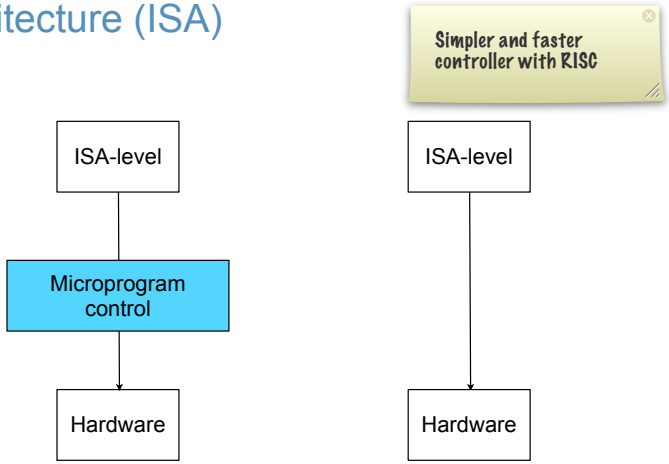
- Limited instruction set with simple instructions
  - speeds up execution, hardwired (goal - 1 instruction per machine cycle)
- Instructions use only few addressing modes
  - register, direct, register indirect, displacement
- Instructions are of fixed length and uniform format
  - ease load and decode, address field at same position
- Load-store architecture (register-to-register operands)
  - without memory reference
  - with memory reference




FI: Fetch Instruction
DI: Decode Instruction
EI: Execute Instruction
CA: Compute Address
TR:Transfer



## Implementation of instruction set architecture (ISA)



## Outline

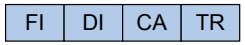
- Control unit
- Input/Output Devices and System Buses
- Programmed I/O, Interrupt-driven I/O, and Direct Memory Access
- RISC and CISC
  - The problem and motivation
  - Register file
  - Instruction set
  - Pipeline



## RISC architecture

- Load-store architecture (register-to-register operands)
  - without memory reference


  - with memory reference

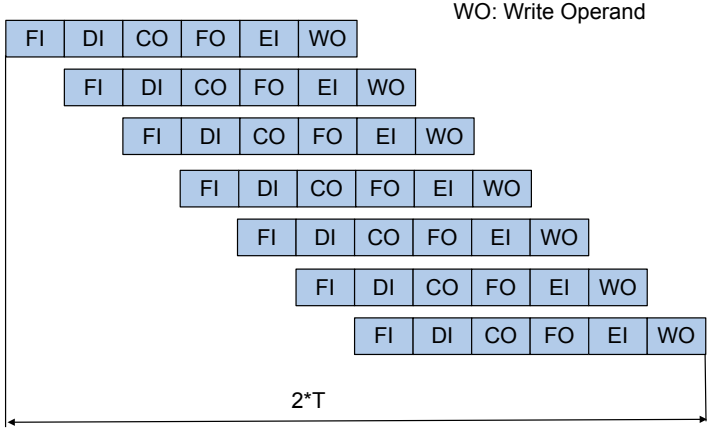



FI: Fetch Instruction
DI: Decode Instruction
EI: Execute Instruction
CA: Compute Address
TR:Transfer



## Pipelining

FI: Fetch Instruction
DI: Decode Instruction
CO: Calculate operand
FO: Fetch Operand
EI: Execute Instruction
WO: Write Operand



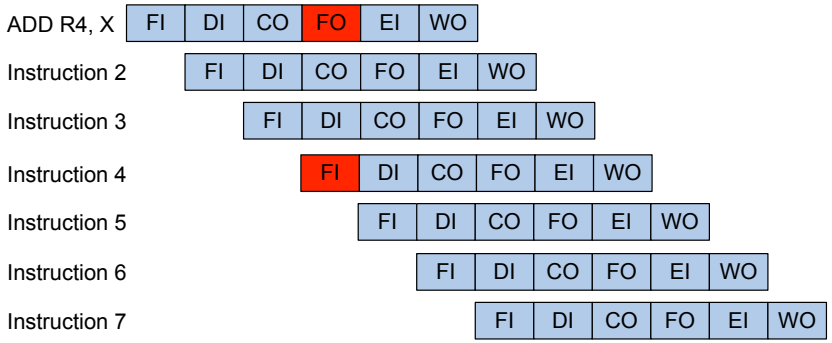



## Pipeline Hazards

- Structural hazards
- Data hazards
- Control hazards

Pipeline hazards prevent the next instruction
The instruction is said to be stalled.
When an instruction is stalled, all instructions later in the pipeline than the stalled instruction are also stalled.
Instructions earlier than the stalled one can continue.
No new instructions are fetched during the stall.

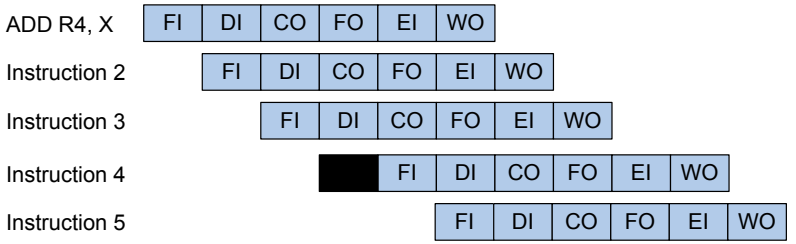
## Structural hazards




Structural hazards occur when a certain resource (memory, functional unit) is requested by more than one instruction at the same time.



## Structural hazards

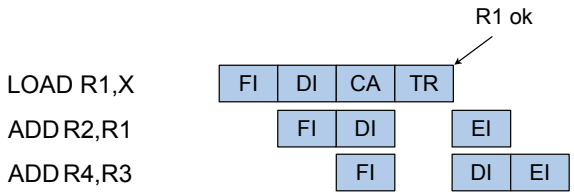



Penalty: 1 cycle

Load/store architecture - only load and store instructions may operate on main memory. Other instructions, such as add, do only operate on registers.



## RISC pipeline - delayed load

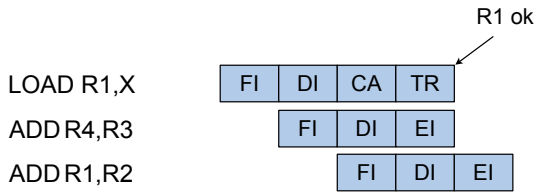



R1 ready after TR
Two alternatives
Delay-stall or delay load



## RISC pipeline - delayed load

load-delay slot



## Comparing RISC and CISC

- Assume a program with 80% simple instructions and 20% complex
- CISC machine (cycle time is 100 ns ($10^{-7}$ s)):
  - simple instructions = 4 cycles
  - complex instructions = 8 cycles
- RISC machine (cycle time is 75 ns ($0.75 * 10^{-7}$ s)):
  - simple instructions = 1 cycle
  - complex operations = sequence of instructions (average 14) = 14 cycles
- How much time takes a program of 1 000 000 instructions?
  - CISC: $(10^6*0.80*4 + 10^6*0.20*8)*10^{-7} = 0.48$ s
  - RISC: $(10^6*0.80*1 + 10^6*0.20*14)*0.75*10^{-7} = 0.27$ s



## Comparing RISC and CISC

- Complex operations take more time on the RISC, but their number is small;
- because of its simplicity, the RISC works at a smaller cycle time; with the CISC, simple instructions are slowed down because of the increased data path length and the increased control complexity.

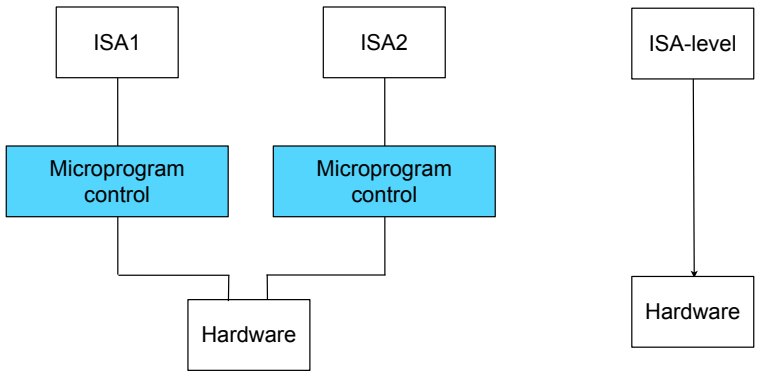


## CISC

- A large number of instructions
- Complex instructions and data types
- Many and complex addressing modes.
- High-level instructions map direct to instructions
- Microprogramming to implement instructions
- Memory bottleneck is a major problem:
  - complex addressing modes and multiple memory
  - accesses per instruction.



## CISC



## CISC

- Advantages:
  - Easier to map high-level instruction to machine instruction
  - Smaller programs; less memory
  - Fewer instructions, lead to smaller execution time.
- Disadvantages
  - A large instruction set is difficult to decode and execute
  - Instructions may not match all high-level language exactly,
  - Complex design tasks.



## CISC processors

- VAX 11/780
  - Nr. of instructions: 303
  - Instruction size: 2 – 57 bytes
  - Instruction format: not fixed
  - Addressing modes: 22
  - Number of general purpose registers: 16
- Pentium
  - Nr. of instructions: 235
  - Instruction size: 1 – 11 bytes
  - Instruction format: not fixed
  - Addressing modes: 11
  - Number of general purpose registers: 8



## CISC - Intel 486

- 32-bit processor
- Registers
  - 8 general
  - 6 address
  - 2 status/control
  - 1 instruction pointer (program counter)
- On-chip floating point unit
- Micro-programmed control
- Instruction set:
  - 253 instructions
  - Instruction size: 1-12 bytes
  - Addressing modes: 11



## RISC

- Limited instruction set
- Simple instructions and data types.
- Few and simple addressing modes
- Instructions are of fixed length
- Load-and-store architecture
- Hardwired controller to implement instructions

Limited instruction set, simple instructions, few addressing modes, and instructions of fixed length make the control unit simpler and faster.

Load/store reduces pipeline penalties

## RISC processors

- Sun SPARC
  - Nr. of instructions: 52
  - Instruction size: 4 bytes
  - Instruction format: fixed
  - Addressing modes: 2
  - Number of general purpose registers: up to 520
- PowerPC
  - Nr. of instructions: 206
  - Instruction size: 4 bytes
  - Instruction format: not fixed (but small differences)
  - Addressing modes: 2
  - Number of general purpose registers: 32



## Summary

- Both RISCs and CISCs try to cover the semantic gap
- CISC approach: implements more and more complex instructions
- RISC approach: try to simplify the instruction set
- Main features of RISC architectures are:
  - reduced number of simple instructions,
  - few addressing modes,
  - load-store architecture,
  - instructions are of fixed length and format,
  - a large number of registers is available.
- One main concerns for RISC - maximize the efficiency of pipelining.
- Present architectures often include both RISC and CISC features.



## RISC Architectures

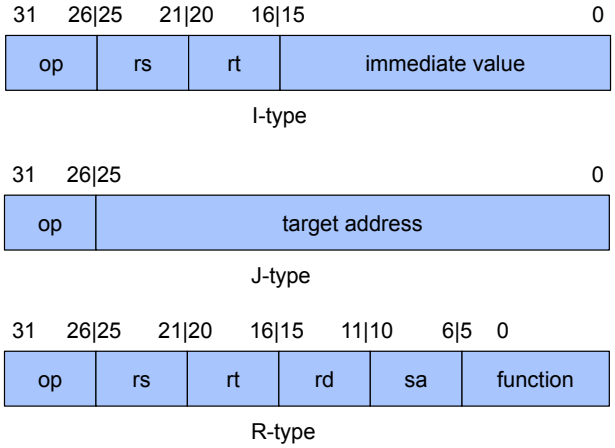
- MIPS
- SPARC
- PowerPC
- ARM



## MIPS

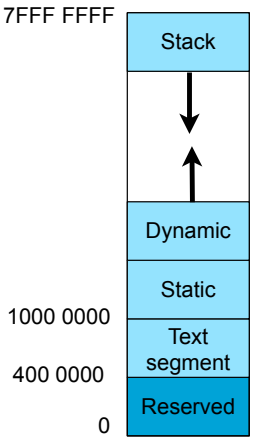
- MIPS(Microprocessor without Interlocked Pipeline Stages)
- MIPS32, 32-bits, MIPS64, 64-bits
- 32 general purpose registers (R0=0, R31=link register), Program counter, 2 register for multiplication/division
- Load/store architecture
- Fixed-length instruction format (32 bits)
  - Immediate (I-type): load and store instructions. The immediate value is 16 bits.
  - Jump (J-type): 26-bit target address is combined with higher-order bits of PC to get absolute address
  - Register (R-type): Arithmetic and logical instructions use the format as well as instructions where the target address is specified indirectly via a register.

## MIPS instruction format






## MIPS memory structure






## SPARC

- Scalable Processor ARCitecture (SPARC) developed by SUN and is based on RISC II from University of California, Berkely.
- Open architecture (license). Different companies makes the processor.
- 64-bit since 1993.
- A user's program sees 32 general purpose registers of 64-bits. r31-r24 are in-registers, r23-r16 are local registers, r15-r8 out registers and r7-r0 are global registers
- 2 addressing modes
  - Register Indirect with Immediate -> address=content of Rx + constant (Rx can be any register and constant is 13-bit displacement)
  - Register Indirect with Index -> address=content of Rx + content of Ry (Rx and Ry can be any register)



## SPARC - instruction set

- Instruction length: 32 bits
- Only load and store access memory
- Opcode (2-bits) - more bits to detail specific opcode
- Arithmetic instructions:
  - Add - add rs1, rs2, rd  rd<-rs1+rs2
  - Mul - mul rs1, rs2, rd  rd<-rs1*rs2 (64 bits times 64 times -> 128 bits)

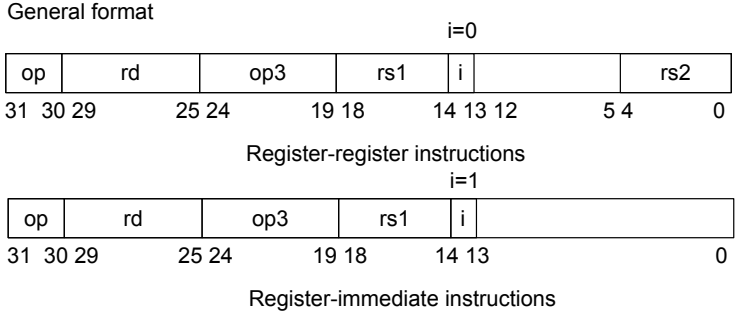
## SPARC - procedure calls

| Caller | Callee | Usage |
|---|---|---|
| %o0 | %i0 | First argument |
| %o1 | %i1 | Second argument |
| %o2 | %i2 | Third argument |
| %o3 | %i3 | Fourth argument |
| %o4 | %i4 | Fifth argument |
| %o5 | %i5 | Sixth argument |
| %o6 | %i6 | Stack pointer |
| %o7 | %i7 | Return adress |



## SPARC - instruction set

General format

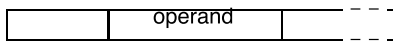





## Immediate addressing

- ADD R4,#3 effect: R4<-R4+3

The operand is directly in one of the fields of the instruction word.

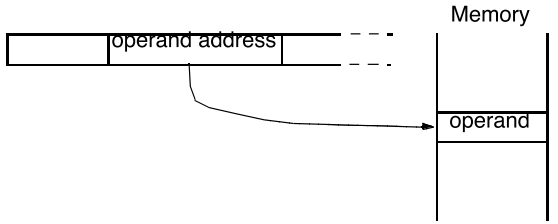





## Direct addressing

- ADD R4,X effect: R4<-R4+[X]

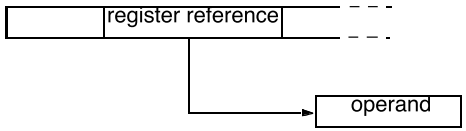
The effective address of the operand is in the instruction word.



## Register addressing

- ADD R4,R3 effect: R4<-R4+R3

Register addressing is similar to direct addressing, but the address field refers to a register rather than main memory.



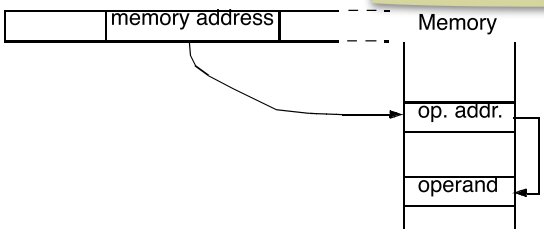



## Memory indirect addressi

- ADD R4,(X) effect: R4<-R4+[[X]]

The instruction word contains the effective address of a memory location which actually contains the effective address of the operand,

With indirect addressing a larger number of memory words can be addressed than with direct addressing

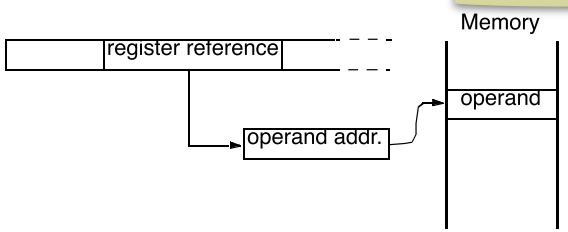





## Register indirect addressing

- ADD R4,(R1) effect: R4<-R4+[R1]

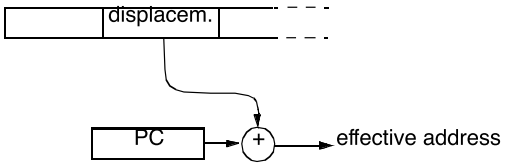
Register indirect addressing is similar to indirect addressing, but the address field refers to a register rather then to main memory.






## Relative addressing

Relative addressing is used in branch instructions. The target of a branch is usually near to the instruction executed -> fewer bits are needed to store the displacement than the effective address of the target instruction.



## SPARC - Window Management

- Up to 32 register windows where each window is 32 registers
- A constant NWINDOWS defines the number of windows
- A pointer CWP (Current Window Pointer) points at the active window

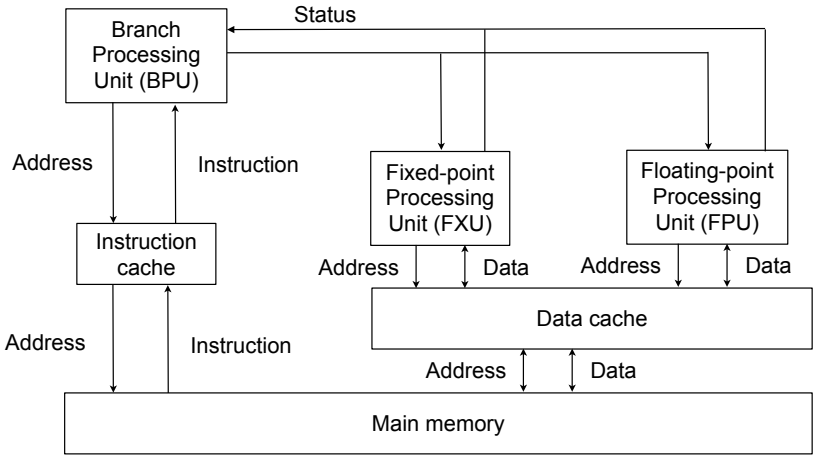



## PowerPC

- PowerPC, developed by IBM. Early attempts from 1975. In early 1990, Motorola, Apple and IBM begun working on PowerPC (power is a RISC instruction set architecture (ISA).
- PowerPC is a 64-bit architecture that can operate in 32-bit or 64-bit mode. Dynamic change between modes. Allows 32-bit binaries (programs) to be executed.



## PowerPC






## PowerPC - register set

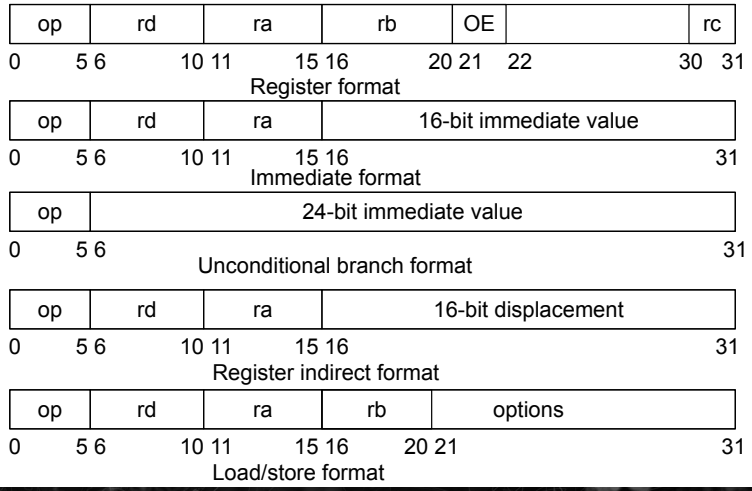
- 32 general purpose registers for integer data
- 32 general purpose registers for floating point data
- 1 condition register - keeps conditions from FXU and FPU
- 1 link register - keeps the return address of procedure calls

## PowerPC - addressing modes

- Register indirect with immediate
  - Effective address = content of rA or 0 + constant
- Register indirect with index
  - Effective address = content of rA + content of rB
- Register indirect with immediate update
  - Effective address = content of rA or 0 + constant
  - rA = effective address
- Register indirect with index update
  - Effective address = content of rA or 0 + content of rB
    - rA = effective address



## PowerPC - instruction set






## ARM

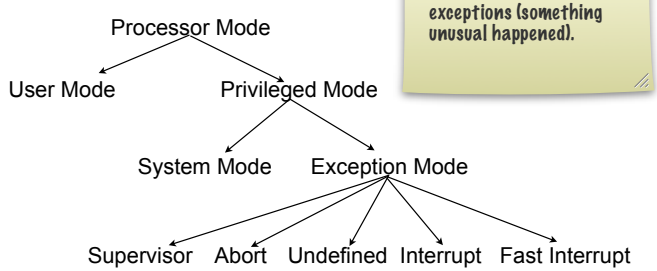
- Acorn RISC Machine (ARM), later Advanced RISC Machine
- Embedded systems such as mobile phones
- First versions had 26-bit address space
- DSP-instructions, Single-Instruction Multiple Data (SIMD) instructions
- 37 registers; 31 general purpose + 6 program status



## ARM - register set

- 16 registers can at any time be accessed by the user, depending on the mode of the processor
- The processor can be in 7 different modes

User programs run in user mode. Privileged mode is for OS or exceptions (something unusual happened).



## ARM

- 9 addressing modes
- 16 conditions possible on each instruction
  - Equal (Z=1), Not Equal (Z=0), Carry (C=1), No Carry (C=0), Negative (N=0), Not negative (N=1), Overflow (V=1), Not overflow (V=0), Unsigned higher (C=1 and Z=0), Unsigned lower (C=0 and Z=1), Signed greater than or equal (N=V), Signed less than (N!=V), Signed greater than (Z=0 and N=V), Signed less than or equal (Z=1 or N!=V), Always, Never



## Summary

- Instruction size: 4 bytes (MIPS, SPARC, PowerPC)
- Instruction sets for PowerPC and ARM are fairly advanced
- ARM has quite many addressing modes



## Questions

- What does the control unit do?
- How can you implement the control unit?
- If execution of an instruction consists of fetch and execute, detail what the control unit should do during fetch.
- What is RISC? CISC? Which to pick?
- What is typical for a CISC (RISC)?
- Name a few RISC processors (and a few CISC processors)
- What is the RISC philosophy to minimize “FO hazards”?
- What type of hazard is a “FO” hazard?
- How can I/O be handled?
- What is interrupt? How does it work?
- What is programmed I/O? Name disadvantages.
- Detail an instruction you would not see in a RISC machine
- Which alternative exists to handle subroutine and procedure calls (which is best from performance (speed) point of view)
- Is fix-length instructions good or bad?