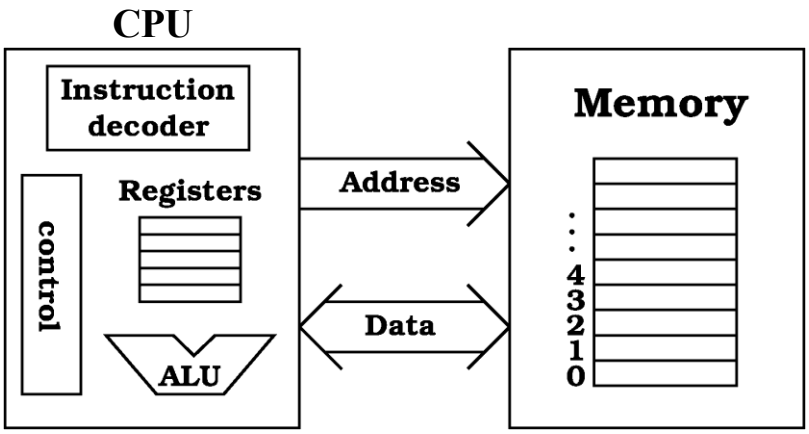
Introduction to Processor Architecture

A simple computer



A simple computer

- The example computer has a Central Processing Unit (CPU), memory (RAM) and 2 busses (Address and Data) that connect them
- The CPU has a set of registers (usually < 100 but may be as few as 4), that are often used to store local operands/variables/intermediate results
- The Arithmetic Logic Unit (ALU) performs computations
- The CPU fetches instructions from the memory where the Instruction Decoder in conjunction with the Control Unit are used to control the elements within the CPU to execute the instructions

A simple computer

- A computer architecture is defined by its instruction set and *architectural state*
- For example, for a 'MIPS' processor the architectural state comprises the program counter (PC) and the 32 registers
- So, based on its current architectural state, the processor executes a particular instruction with a particular set of data to yield a new architectural state
- The *microarchitecture* is the specific arrangement of registers, ALUs, finite state machines (FSMs), memories and other logic building blocks (e.g., multiplexers) needed to implement an architecture
- Note that a particular architecture can be implemented by many different microarchitectures, each having different performance, complexity and cost trade-offs

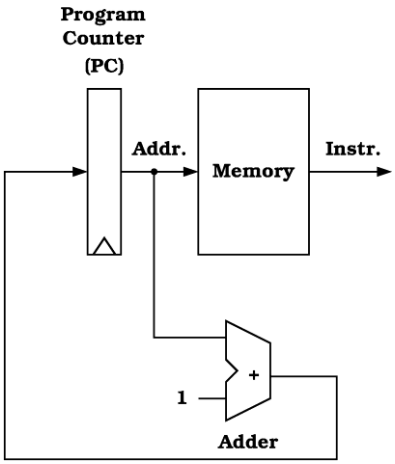
Microarchitecture

- A *microarchitecture* can usually be divided in to 2 interacting parts:
  - *Datapath*: Operates on words of data, e.g., 16-bit, 32-bit, and contains structures such as memories, , registers, ALUs and multiplexers. Note that the program counter can be viewed as a conventional register whose output points to the current instruction and its input indicates the address of the next instruction.
  - *Instruction Decoder/Control Unit*: receives the current instruction from the - and tells the data-path how to execute that instruction, i.e., the control unit issues multiplexer select, register enable and memory write signals to control the operation of the data-path.

Building a simple computer

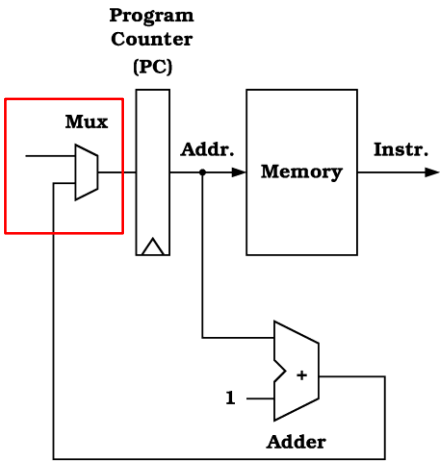
- We will now look the design of a single-cycle processor, i.e., it executes its instructions in a *single* clock cycle
- We gradually develop the data-path by adding new components to the state elements. In doing so, we gradually increase the capability of the computer
- The instruction decoder/control unit generates the control signals (using combinational logic) that control the data-path so that the required instructions can be executed
- We will assume that the computer is based on word addressable memory, e.g., 32-bit words at each memory location (address)

Building a simple computer – fetching instructions



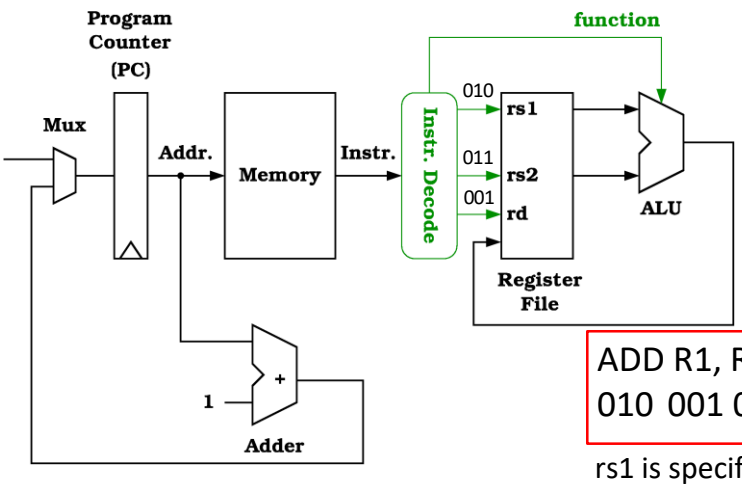
- Address supplied from PC to memory yields instruction to be executed
- This instruction is presented to the rest of the data path
- PC is then incremented by one (utilising the Adder) to point to the next instruction in the memory
- Can't write very interesting programs!
  - No branching
  - No access to data in memory

Building a simple computer – branching enable



- Including Mux enables PC to be changed to an arbitrary value to permit branching – detail to follow
- We will now introduce more of the data-path
  - Instruction decoder/control unit
  - Registers (actually register file) and ALU
- We will return to branching later!

Building a simple computer – register access



Example machine instruction format

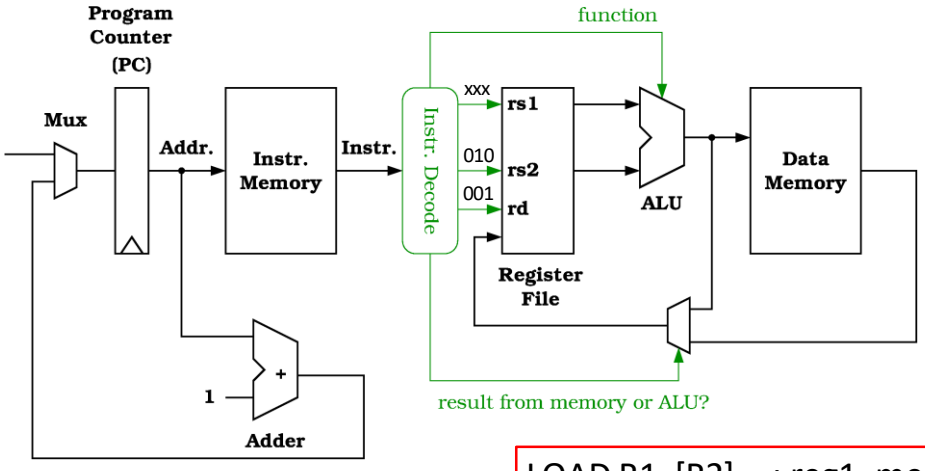
op	rd	rs1	rs2

op – operation code
rs1 – ALU source register
rs2 – ALU source register
rd – ALU destination register

ADD R1, R2, R3 ; reg1 = reg2 + reg3
010 001 010 011

rs1 is specified as R2, rs2 is specified as R3,
rd is specified as R1 and ALU function - ADD

Building a simple computer – memory access

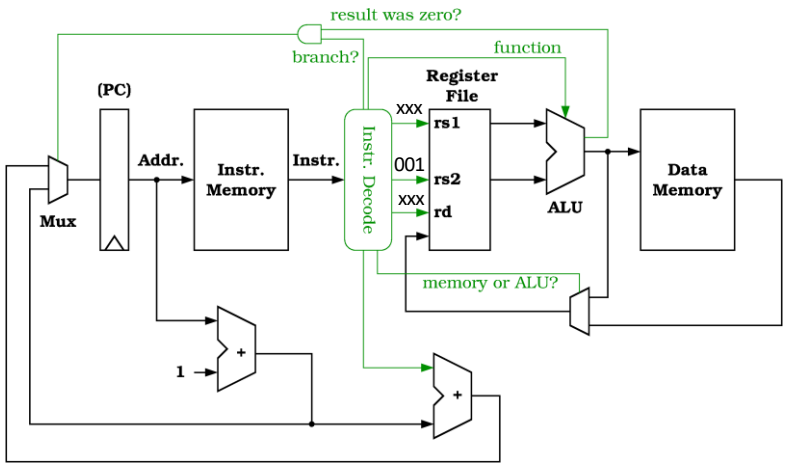


LOAD R1, [R2] ; reg1=mem[reg2]

Building a simple computer – memory access

- Addition of Mux permits result to be stored in the Register File (at destination rd specified as R1 (reg1)) to come from ALU or from Data Memory
- Data memory address specified by ALU output
- ALU input/output is content of source register rs2 specified as R2 (reg2)
- So Data Memory output is the content of the location pointed to by R2 (reg2)
- That is LOAD R1, [R2] ; reg1=mem[reg2]
- Note source register rs1 is not required in this operation and so does not need to be specified

Building a simple computer - branching



BEQZ R1, +10 ; if (R1=0) PC=PC+10

Building a simple computer - branching

- ALU input/output is the contents of source register rs2 specified as R1 (reg1)
- ALU also has a flag output indicating if the ALU output is zero
- If a branch instruction is decoded (by the Instruction decoder/controller) and the ALU zero flag is set, then the AND gate output (which is the branch Mux control input) will become '1' and the input to the PC will now come from the output of the newly introduced 'jump' adder
- The jump adder takes the current PC value and adds to it the required 'jump' value (supplied by the Instruction decoder)

Building a simple computer - branching

- For example, for the branch if equal to zero instruction

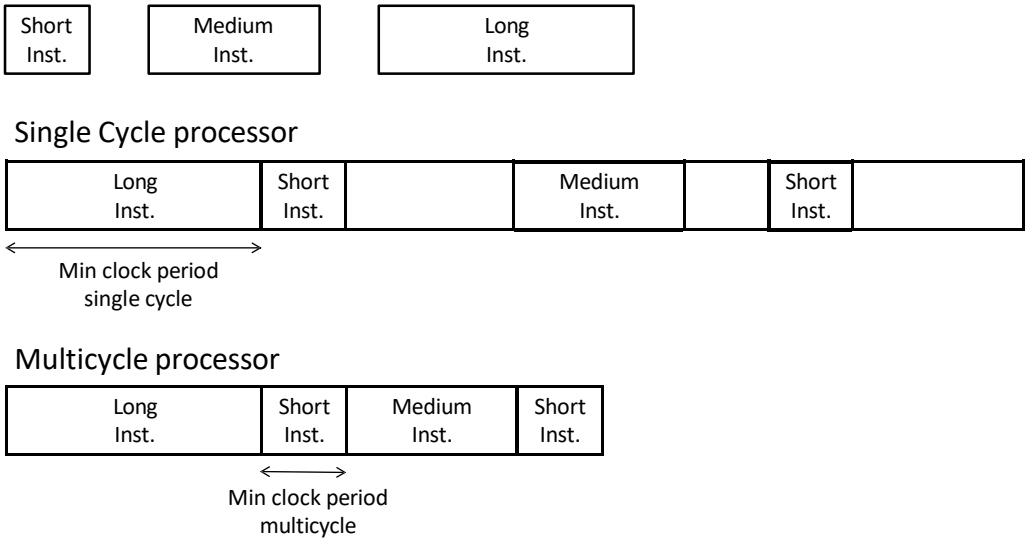
```
BEQZ R1, +10 ; if (reg1=0) PC=PC+10
```

- When executed a jump of 10 instructions will occur (i.e., 10 is added to the PC) if the contents of register specified as R1 (reg1) is equal to zero

Multicycle processor

- A single cycle processor has 3 main weaknesses:
  - Clock cycle needs to be long enough to cope with slowest instruction
  - Needs 3 adders – 1 in ALU and 2 in the PC logic
  - Separate instruction and data memory
- In a multicycle processor:
  - Instructions are broken into multiple shorter (i.e., faster) steps
  - More complex instructions take more steps than simple ones, so simple instructions execute faster than complex ones
  - Need only one adder since this can be reused for different tasks in different steps
  - Only one memory is required since instruction is fetched in 1st step and data may be read or written in later steps

Multicycle processor



Multicycle processor

- Design is more complex:
  - Need to add non-architectural state elements (i.e., registers) to hold intermediate results
  - The controller is now a FSM rather than combinational logic since it has to produce different outputs on different steps
- Advantages:
  - ALU can now be reused several times
  - Instructions and data can be stored in one shared memory (since memory accesses are now separate)

Execution time

Time/program = instruction count x av. time to execute an instruction

Time/instruction = clocks per instruction x clock period

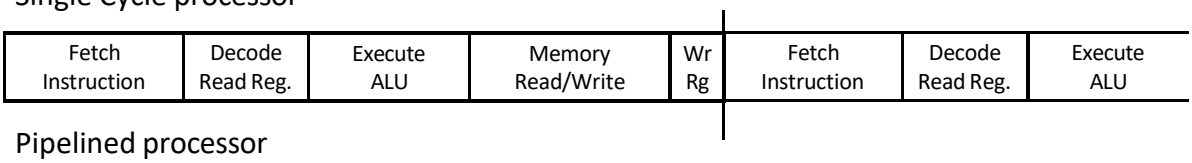
How do we build a fast computer?

Pipelined processor

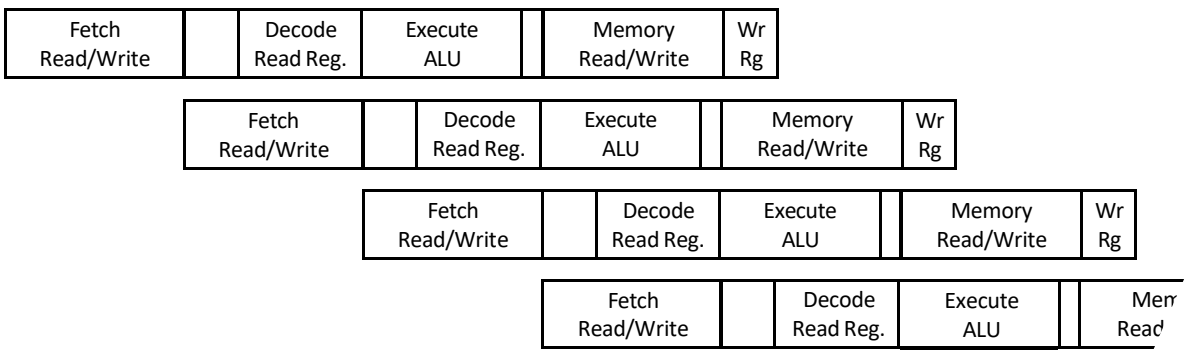
- In a similar way to that used in a multicycle processor, instructions are broken up into say, 5 smaller steps, e.g., fetch, decode, execute ALU, memory read/write, write register
- Since each stage is less complex it will execute about 5 times faster
- In this case, dividing the single cycle processor in to 5 'pipelined' stages means that 5 instructions can execute simultaneously, one in each stage, i.e., the throughput is ideally 5 times greater, i.e., compared with a conventional single cycle processor, i.e., a fetch occurs every clock cycle in a pipelined processor compared with once every instruction in a conventional single cycle processor

Pipelined processor

Single Cycle processor



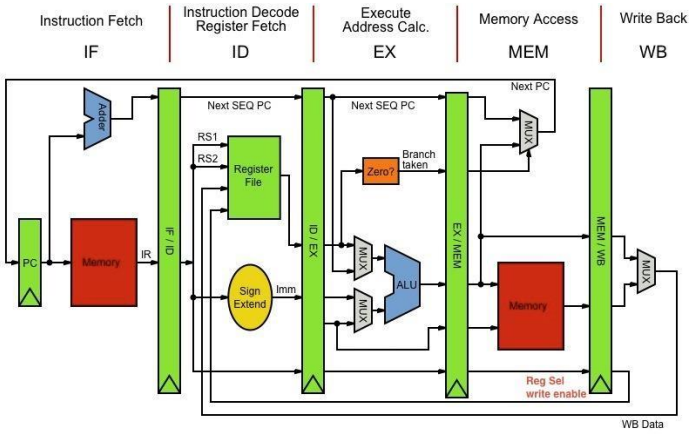
Pipelined processor



Pipelined processor

- Note that the register file is written in the 1st part of a cycle and read in the second part so that data can be written and read back within a single cycle
- The central challenge is handling hazards, i.e., when the results from one instruction are needed by a subsequent instruction before the former has completed
- Actually, there are 2 kinds of hazards
  - Data hazard – when an instruction tries to read a register that has not yet been written back
  - Control hazard – when the decision of what instruction to fetch next has not been made by the time the fetch takes place
- These issues will be addressed in the Computer Architecture course!

Pipelined processor



DDL Representation

DDL Representation is a structured way of showing the CPU datapath and control signals to explain how data flows inside the processor during instruction execution. It visually represents components such as registers, ALU, multiplexers, memory, and control units, along with their interconnections. This makes it easier to understand how instructions are fetched, executed, and completed. DDL is commonly used to compare single-cycle, multi-cycle, and pipelined processors and is an important tool for teaching CPU design and understanding processor behavior.

Exceptions

An exception occurs when the normal execution of a program is interrupted due to an abnormal condition such as divide by zero, invalid instruction, overflow, or memory access error. When an exception happens, the CPU saves its current state and transfers control to the operating system's exception handler to manage the error. In pipelined processors, handling exceptions is more complex because multiple instructions are in progress, so the pipeline may need to be flushed. Exceptions are essential for maintaining system stability, correctness, and security.