ISA and Sequential CPU

CPSC 313 M1

Narendra Syahrasyad

1 Processing in Stages

Processing an instruction involves a number of different operations.

1. **Fetch**

   Reads bytes of an instruction from memory. Parses the instruction into several variables:

   (a) `icode` - instruction code

   (b) `ifun` - instruction function

   (c) `rA` - first register operand, if exists

   (d) `rB` - second register operand, if exists

   (e) `valC` - 4 byte word included in instruction, if exists

   Fetch computes an additional variable `valP`, which is equal to `PC` + length of fetched instruction.

2. **Decode**

   Reads values from register file into `valA` and `valB`. Typically reads from registers `rA` and `rB`, but sometimes from other registers.

3. **Execute**

   ALU either performs the operation specified by the instruction, computes address of a memory reference, or manipulates the stack pointer. The result is saved onto `valE`.

   Condition codes are set afterwards. Jump instructions use these codes to determine if a branch should be taken.

4. **Memory**

   Reads/writes data from/to memory. Value read saved as `valM`.

5. **Write back**

   Writes up to two results to the register file.

6. **PC update**

   The PC is set to the address of the next instruction.

1.1 Condition Codes

There are three condition codes:

- `ZF` Zero Flag: Indicates if the result `valE` was zero
- `SF` Sign Flag: Indicates if the result was negative
- `OF` Overflow Flag: Indicates if the result of the operation did not fit in the register width

1.2 Execution of OPq

All `OPq` operations follow the same format:

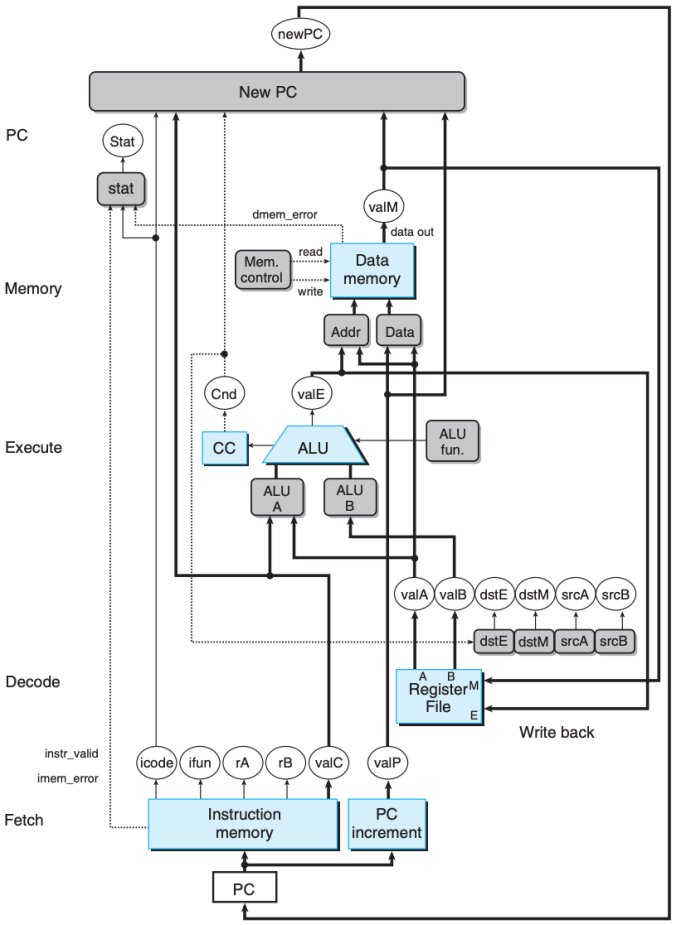
Stage	Generic
Fetch	`icode:iFun` $\leftarrow M_1$[PC]
	`rA:rB` $\leftarrow M_1$[PC + 1]
	`valP` $\leftarrow$ PC + 2
Decode	`valA` $\leftarrow$ R[rA]
	`valB` $\leftarrow$ R[rB]
Execute	`valE` $\leftarrow$ `valB` OP `valA`
	set condition codes
Memory	
Write back	R[rB] $\leftarrow$ `valE`
PC update	PC $\leftarrow$ `valP`

1.3 Jump/Call/Ret Instructions

In `Y86` ISA the stack grows **down**.

- The `call` instruction decrements the stack pointer and saves the return address to the first element of the stack.
- The `ret` instruction increments the stack pointer and gets the return address from the top of the stack.

2 Hardware Structure



The different hardware units are associated with different processing stages.

1. **Fetch**

   Using the `PC` register at the bottom, the instruction memory reads and parses the bytes of the instruction. The PC increment unit computes `valP`.

2. **Decode**

   The register file uses its A and B ports to read the values of `rA` and `rB` to `valA` and `valB`.

3. **Execute**

   The ALU computes the appropriate value according to the instruction type. The condition code register (CC) contains all condition codes. It computes `Cnd` which is used for all conditional instructions.

4. **Memory**

The data memory reads/writes a word of memory when executing a memory instruction.

5. **Write back**

   Writes back to register file through ports E and M. Port E is for values calculated by ALU, port M is for values read from data memory.