Sequential Y86-64 Implementations

Team 12

Eswara Rohan (2020102002)

Keerthi Pothalaraju (2020102010)

Objective –

To build a sequential Y86-64 processor. (SEQ)

Description –

Here we build a Y86-64 processor in stages. On each clock cycle, SEQ performs all the steps required to process a complete instruction.

The steps and operations performed on each step are described below –

1. Fetch - Read instruction from instruction memory.
2. Decode - Read program registers
3. Execute - Compute value or address
4. Memory - Read or write back data.
5. Write Back - Write program registers.
6. PC Update - Update the program counter

Sequential Y86-64 Implementations

Fetch –

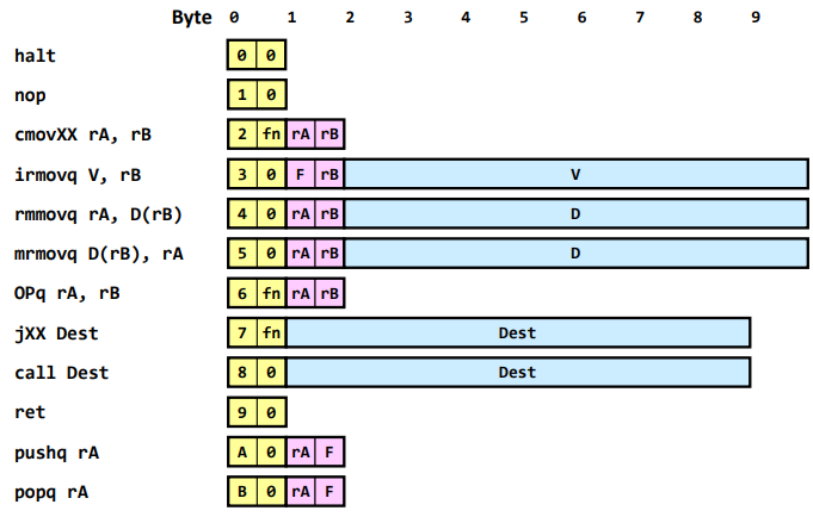
This stage reads the bytes of an instruction from memory using Program Counter (PC) as the memory address.

Computed Values in this stage are -

icode – Instruction Code
ifun – Function Code
rA – Inst. Register A
rB – Inst. Register B
valC – Instruction Constant
valP – Incremented Program Counter

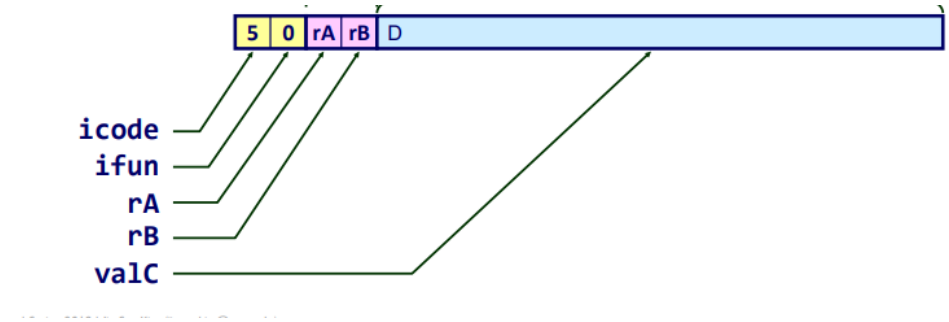
Implementation Of Fetch Stage -

Below is **Y86-64 Instruction Set -**



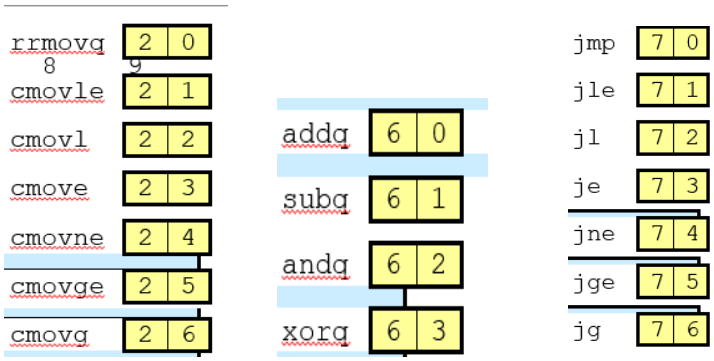
Sequential Y86-64 Implementations

Below is the method to determine icode, ifun, rA, rB and valC of a given instruction.



Based on the above two figures we can determine icode, ifun, rA, rB, valC values.

For determination of ifun in case of cmovXX, OPq, jXX. -



Calculation Of valP -

valP can be determined from icode -

For **halt, nop, ret** - valP = PC + 64'd1
For **cmovXX, OPq, pushq, popq** - valP = PC + 64'd2
For **call, jXX** - valP = PC + 64'd9
For **irmovq,rmmovq,mrmovq** - valP = PC + 64'd10

Sequential Y86-64 Implementations

Decode and Write-Back –

Decode reads the registers designated by rA and rB and output values valA and valB but for some instructions it reads register %rsp.
Write-Back write program registers.

Computed Values in this stage are -

valA - Register Value A
valB - Register Value B

Implementation Of Decode and Write Back Stage -

OPq	Decode	valA ← R[rA]	Read operand A
rmmovq	Decode	valA ← R[rA]	Read operand A
mrmovq	Decode		
irmovq	Decode		
pushq	Decode	valA ← R[rA]	Read operand A
popq	Decode	valA ← R[%rsp]	Read stack pointer
cmovXX	Decode	valA ← R[rA]	Read operand A
jXX	Decode		
call	Decode		
ret	Decode	valA ← R[%rsp]	Read stack pointer

OPq	Write Back	R[rB] ← valE	Write back result
rmmovq	Write Back		
mrmovq	Write Back		
irmovq	Write Back	R[rB] ← valE	Write back result
pushq	Write Back	R[%rsp] ← valE	Update stack pointer
popq	Write Back	R[%rsp] ← valE	Update stack pointer
cmovXX	Write Back	R[rB] ← valE	Write back result
jXX	Write Back		
call	Write Back	R[%rsp] ← valE	Update stack pointer
ret	Write Back	R[%rsp] ← valE	Update stack pointer

Sequential Y86-64 Implementations

Execute –

This stage performs either of the following two actions -

a) ALU performs the operation specified by ifun and computes effective address of memory.
b) Increments (or) Decrements the stack pointer.

Computed Values in this stage are -

valE - ALU Result
Cnd - Constant to determine whether to take a branch or not.

Implementation Of Execute Stage -

OPq	Execute	valE ← valB OP valA	Perform ALU operation
rmmovq	Execute	valE ← valB + valC	Compute effective address
mrmovq	Execute	valE ← valB + valC	Compute effective address
irmovq	Execute	valE ← valB + valC	Pass valC through ALU
pushq	Execute	valE ← valB + (-8)	Decrement stack pointer
popq	Execute	valE ← valB + 8	Increment stack pointer
cmovXX	Execute	valE ← valB + valA	Pass valA through ALU
jXX	Execute		
call	Execute	valE ← valB + (-8)	Decrement stack pointer
ret	Execute	valE ← valB + 8	Increment stack pointer

In case of cmovXX, to determine Cnd value we use the following conditions -

Sequential Y86-64 Implementations

Instruction		Synonym	Move condition	Description
`cmove`	*S*, *R*	`cmovz`	`ZF`	Equal / zero
`cmovne`	*S*, *R*	`cmovnz`	`~ZF`	Not equal / not zero
`cmovs`	*S*, *R*		`SF`	Negative
`cmovns`	*S*, *R*		`~SF`	Nonnegative
`cmovg`	*S*, *R*	`cmovnle`	`~(SF ^ OF) & ~ZF`	Greater (signed >)
`cmovge`	*S*, *R*	`cmovnl`	`~(SF ^ OF)`	Greater or equal (signed >=)
`cmovl`	*S*, *R*	`cmovnge`	`SF ^ OF`	Less (signed <)
`cmovle`	*S*, *R*	`cmovng`	`(SF ^ OF) | ZF`	Less or equal (signed <=)
`cmova`	*S*, *R*	`cmovnbe`	`~CF & ~ZF`	Above (unsigned >)
`cmovae`	*S*, *R*	`cmovnb`	`~CF`	Above or equal (Unsigned >=)
`cmovb`	*S*, *R*	`cmovnae`	`CF`	Below (unsigned <)
`cmovbe`	*S*, *R*	`cmovna`	`CF | ZF`	Below or equal (unsigned <=)

We use similar conditions in case of jXX to determine the value of Cnd.

Memory -

Memory either read data from memory or write data to memory.

Computed Values in this stage are -

valM - Value read from memory

Implementation Of Memory Stage -

OPq	Memory		
rmmovq	Memory	M_8[valE] ← valA	Write value to memory
mrmovq	Memory	valM ← M_8[valE]	Read value from memory
irmovq	Memory		
pushq	Memory	M_8[valE] ← valA	Write to stack
popq	Memory	valM ← M_8[valA]	Read from stack
cmovXX	Memory		
jXX	Memory		
call	Memory	M_8[valE] ← valP	Update stack pointer
ret	Memory	valM ← M_8[valA]	Update stack pointer

In case of rmovq, call and pushq we write to memory. Whereas, in case of mrmovq, ret and popq we read from memory.

Sequential Y86-64 Implementations

PC Update

New value of the PC is taken in one of valC, valM, valP.

Computed Values in this stage are -

PC__Update - Updated Program Counter

Implementation Of PC Update Stage -

OPq	PC Update	PC ← valP	Update PC
rmmovq	PC Update	PC ← valP	Update PC
mrmovq	PC Update	PC ← valP	Update PC
irmovq	PC Update	PC ← valP	Update PC
pushq	PC Update	PC ← valP	Update PC
popq	PC Update	PC ← valP	Update PC
cmovXX	PC Update	PC ← valP	Update PC
jXX	PC Update	PC ← Cnd? valC : valP	Update PC
call	PC Update	PC ← valC	Update PC
ret	PC Update	PC ← valM	Update PC

Sequential Y86-64 Implementations

Important points:

- In the Y86 processor implementation there are 5 stages.
    1. Fetch
    2. Decode
    3. Execute
    4. Memory
    5. Write back
    6. PC update

```
Instruction_memory[0] = 8'b01100000; //6 add
Instruction_memory[1] = 8'b00000011; //%rax %rbx and store in rbx

Instruction_memory[2] = 8'b00100000; // rrmovq
Instruction_memory[3] = 8'b00000011; // src = %rax dest = %rdx



Instruction_memory[4] = 8'b01000000; //4-rmmovq
Instruction_memory[5] = 8'b00000011; //rax and (rbx)
Instruction_memory[6] = 8'b00000000;
Instruction_memory[7] = 8'b00000000;
Instruction_memory[8] = 8'b00000000;
Instruction_memory[9] = 8'b00000000;
Instruction_memory[10] = 8'b00000000;
Instruction_memory[11] = 8'b00000000;
Instruction_memory[12] = 8'b00000000;
Instruction_memory[13] = 8'b00001111;


Instruction_memory[14] = 8'b00010000; //no operation
Instruction_memory[15] = 8'b00010000; //no operation
Instruction_memory[16] = 8'b00000000; //halt
```

Sequential Y86-64 Implementations

```
clk = 1 icode = 0110 ifun = 0000  rA = 0000 rB = 0011 valA =                  20 valB =                   1  valC=                   0 valE=                  21 valM=                   0 Halt_Prog=0 In
structionValid=1 pcvalid=0 cnd=0
RegMem0=                  20
RegMem1=                  21
RegMem2=                  54
RegMem3=                   1
RegMem4=                  31
RegMem5=                  63
RegMem6=                  12
RegMem7=                  95
RegMem8=                  72
RegMem9=                  52
RegMem10=                  3
RegMem11=                  8
RegMem12=                  9
RegMem13=                  4
RegMem14=                  6
ZF=0
SF=0
OF=0
clk = 0 icode = 0110 ifun = 0000  rA = 0000 rB = 0011 valA =                  20 valB =                  21  valC=                   0 valE=                  21 valM=                   0 Halt_Prog=0 In
structionValid=1 pcvalid=0 cnd=0
RegMem0=                  20
RegMem1=                  21
RegMem2=                  54
RegMem3=                  21
RegMem4=                  31
RegMem5=                  63
RegMem6=                  12
RegMem7=                  95
RegMem8=                  72
RegMem9=                  52
RegMem10=                  3
RegMem11=                  8
RegMem12=                  9
RegMem13=                  4
RegMem14=                  6
ZF=0
SF=0
OF=0
```

```
clk = 1 icode = 0010 ifun = 0000  rA = 0000 rB = 0011 valA =                  20 valB =                  21  valC=                   0 valE=                  20 valM=                   0 Halt_Prog=0 In
structionValid=1 pcvalid=0 cnd=1
RegMem0=                  20
RegMem1=                  21
RegMem2=                  54
RegMem3=                  21
RegMem4=                  31
RegMem5=                  63
RegMem6=                  12
RegMem7=                  95
RegMem8=                  72
RegMem9=                  52
RegMem10=                  3
RegMem11=                  8
RegMem12=                  9
RegMem13=                  4
RegMem14=                  6
ZF=0
SF=0
OF=0
clk = 0 icode = 0010 ifun = 0000  rA = 0000 rB = 0011 valA =                  20 valB =                  20  valC=                   0 valE=                  20 valM=                   0 Halt_Prog=0 In
structionValid=1 pcvalid=0 cnd=1
RegMem0=                  20
RegMem1=                  21
RegMem2=                  54
RegMem3=                  20
RegMem4=                  31
RegMem5=                  63
RegMem6=                  12
RegMem7=                  95
RegMem8=                  72
RegMem9=                  52
RegMem10=                  3
RegMem11=                  8
RegMem12=                  9
RegMem13=                  4
RegMem14=                  6
ZF=0
SF=0
OF=0
```

```
The memory contained all 3 earlier but now memory[valE] has changed to                  20 in the following

clk = 1 icode = 0100 ifun = 0000  rA = 0000 rB = 0011 valA =                  20 valB =                  20  valC=                  15 valE=                  35 valM=                   0 Halt_Prog=0 In
structionValid=1 pcvalid=0 cnd=0
RegMem0=                  20
RegMem1=                  21
RegMem2=                  54
RegMem3=                  20
RegMem4=                  31
RegMem5=                  63
RegMem6=                  12
RegMem7=                  95
RegMem8=                  72
RegMem9=                  52
RegMem10=                  3
RegMem11=                  8
RegMem12=                  9
RegMem13=                  4
RegMem14=                  6
ZF=0
SF=0
OF=0
clk = 0 icode = 0100 ifun = 0000  rA = 0000 rB = 0011 valA =                  20 valB =                  20  valC=                  15 valE=                  35 valM=                   0 Halt_Prog=0 In
structionValid=1 pcvalid=0 cnd=0
RegMem0=                  20
RegMem1=                  21
RegMem2=                  54
RegMem3=                  20
RegMem4=                  31
RegMem5=                  63
RegMem6=                  12
RegMem7=                  95
RegMem8=                  72
RegMem9=                  52
RegMem10=                  3
RegMem11=                  8
RegMem12=                  9
RegMem13=                  4
RegMem14=                  6
ZF=0
SF=0
OF=0
```

Sequential Y86-64 Implementations

```
clk = 1 icode = 0001 ifun = 0000  rA = 0000 rB = 0011 valA =                0 valB =                0  valC=                0 valE=               35 valM=                0 Halt_Prog=0 In
structionValid=1 pcvalid=0 cnd=0
RegMem0=                   20
RegMem1=                   21
RegMem2=                   54
RegMem3=                   20
RegMem4=                   31
RegMem5=                   63
RegMem6=                   12
RegMem7=                   95
RegMem8=                   72
RegMem9=                   52
RegMem10=                   3
RegMem11=                   8
RegMem12=                   9
RegMem13=                   4
RegMem14=                   6
ZF=0
SF=0
OF=0
clk = 0 icode = 0001 ifun = 0000  rA = 0000 rB = 0011 valA =                0 valB =                0  valC=                0 valE=               35 valM=                0 Halt_Prog=0 In
structionValid=1 pcvalid=0 cnd=0
RegMem0=                   20
RegMem1=                   21
RegMem2=                   54
RegMem3=                   20
RegMem4=                   31
RegMem5=                   63
RegMem6=                   12
RegMem7=                   95
RegMem8=                   72
RegMem9=                   52
RegMem10=                   3
RegMem11=                   8
RegMem12=                   9
RegMem13=                   4
RegMem14=                   6
ZF=0
SF=0
OF=0
```