Y86 ISA

Moving forward, we will use the Y86 instruction set architecture to assist with our discussion of processor architecture. Y86 is a subset of IA32 and it introduces many elements of the RISC paradigm which helps simplify our study of processor architecture.

State in Y86
State in Y86 ISA is maintained through the following:

- 8 general purpose registers
- 3 condition codes (ZF,SF,OF) (zero, negative, overflow)
- The program counter (PC)
- Current program status, STAT (for exception handling)
- Memory, DMEM

Registers

Each register has a corresponding code used for encoding the register in the binary machine language.

Register Name	Register Code
%eax	0
%ecx	1
%edx	2
%ebx	3
%esp	4
%ebp	5
%esi	6
%edi	7

*We designate register code F to mean the null or no register

Instruction Encodings
Each instruction in Y86 has a binary encoding that ranges between 1 and 6 bytes. Each instruction consists of a 1-byte instruction specifier, an optional 1 byte register specifier and an optional 4-byte word constant with *little endian* byte ordering.

	Byte					
Instruction	**0**	**1**	**2**	**3**	**4**	**5**

halt	00	-	-	-	-	-
nop	10	-	-	-	-	-
rrmovl rA,rB	20	rA rB	-	-	-	-
irmovl V, rB	30	F rB	Immediate Value V			
rmmovl rA, D(rB)	40	rA rB	Offset value D			
mrmovl D(rB), rA	50	rA rB	Offset value D			
OP rA, rB	6 fn	rA rB	-	-	-	-
jXX Dest	7 fn	Program location Dest				-
cmovXX rA, rB	2 fn	rA rB	-	-	-	-
call Dest	80	Program destination Dest				-
ret	90	-	-	-	-	-
pushl rA	A0	rA F	-	-	-	-
popl rA	B0	rA F	-	-	-	-

ALU operations

Operation	**Function Code(fn)**
addl	0
subl	1
andl	2
xorl	3

Branch operations

Branch	**Function Code(fn)**
jmp	0
jle	1
jl	2
je	3
jne	4

jge	5
jg	6

Conditional Moves

Conditional Move	**Function Code(fn)**
cmovle	1
cmovl	2
cmove	3
cmovne	4
cmovge	5
cmovg	6

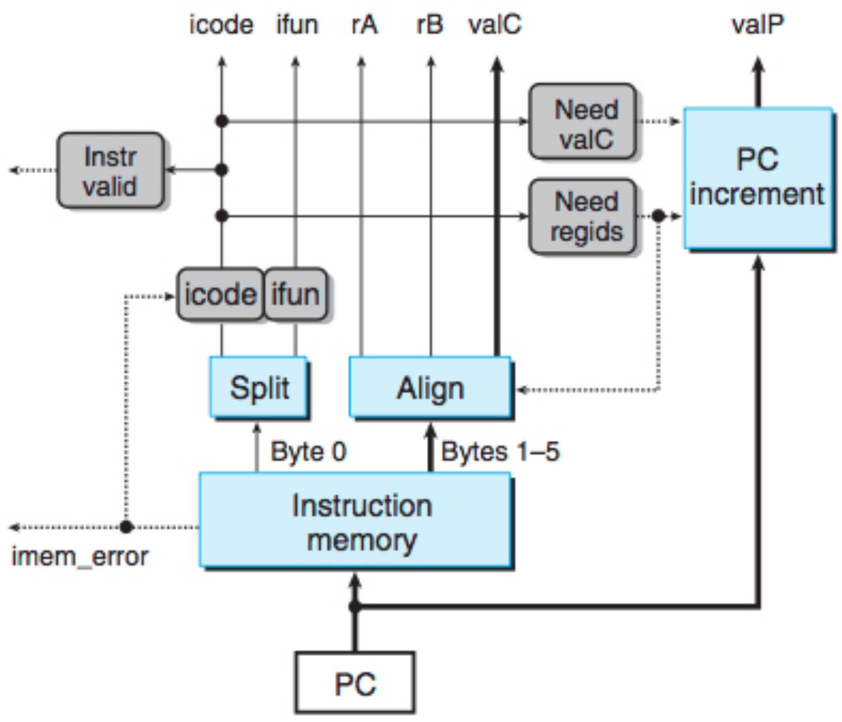
SEQ

We will define SEQ as a "sequential" processor that completes every phase of the instruction cycle in one clock cycle. With each instruction, the following happens during the instruction cycle:

Fetch

- Instruction bytes are read from memory based on the PC.
- the two four-bit instruction (icode) and function (ifun) codes are interpreted.
- Depending on the instruction, one or more register operand specifiers are loaded (rA, rB).
- Depending on the instruction, loads a 32-bit constant value, valC.
- The address of the next instruction, valP is computed.

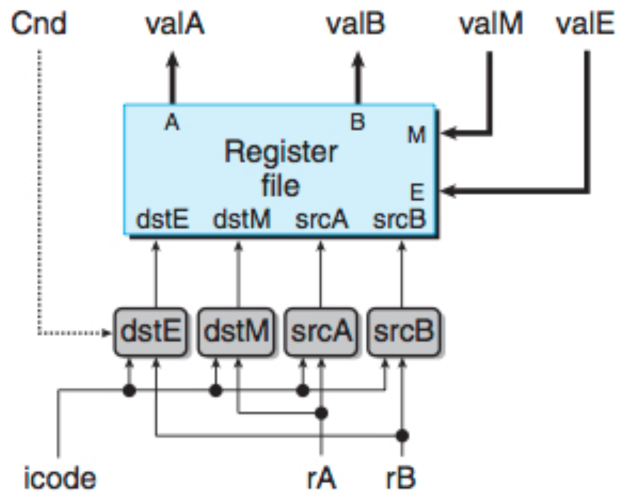
Control diagram for Fetch:



Decode

- Depending on the instruction, reads up to two values valA and valB from the register file based on rA,rB in the instruction field.
- Depending on the instruction, reads %esp register

Control diagram for Decode (and Write-back):

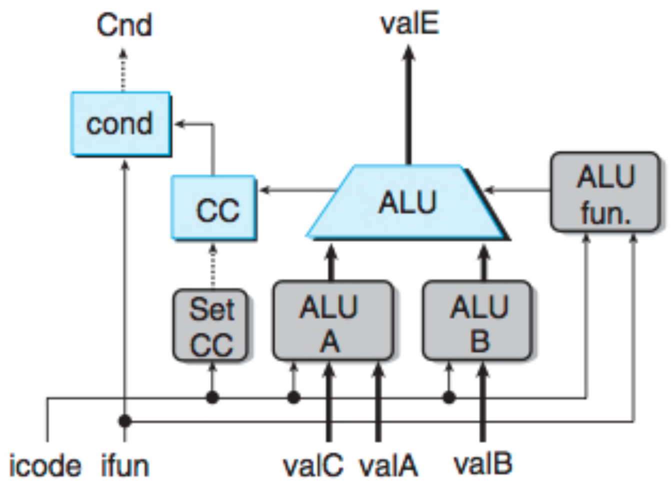


Execute

- ALU performs operation defined by instruction, valE
- ALU computes effective address of a memory reference, valE
- ALU increments or decremnts the stack pointer, valE
- Condition codes are set.

- For jump instructions, condition codes and branch conditions are evaluated, Cnd

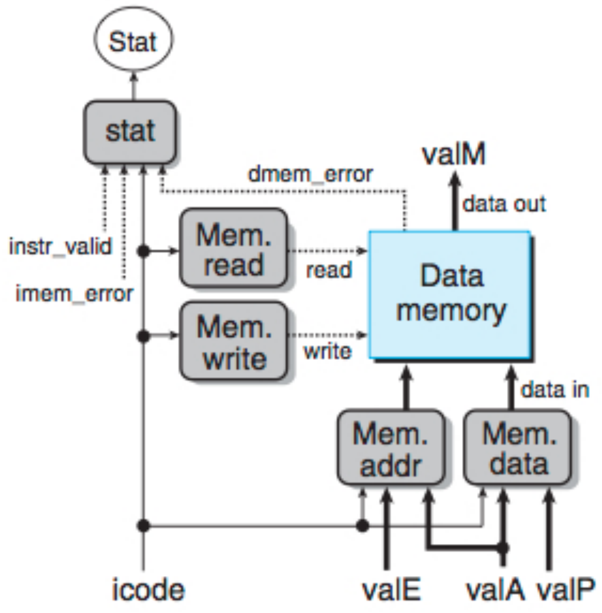
Control diagram for Execute:



Memory

- May write data to memory
- May read value from memory, valM

Contorl diagram for Memory



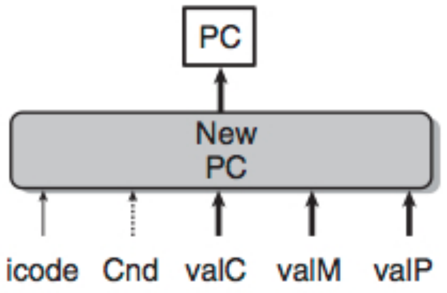
Write-back

- Writes up to two results to the register file.

PC Update

- PC is updated to the address of the next instruction

Control diagram for PC Update:



<u>Example:</u>
Given the following Y86 code, trace the events of each phase of the instruction cycle:

```
0x000               irmovl        $9, %edx
0x006               irmovl        $21, %ebx
0x00C               subl          %edx, %ebx
0x00E               irmovl        $128, %esp
0x014               rmmovl        %esp, 100(%ebx)
0x01A               pushl         %edx
0X01C               popl          %eax
0X01E               je            done
0X023               call          proc
0X028               done:
0X028               halt
0X029               proc:
0X029               ret


0x000:30F209000000            irmovl          $9, %edx

FETCH:
instruction bytes are read, two 4-bit codes are interpreted:
        icode:ifun <- M_1[PC]
        icode:ifun  <- M_1[0x000]
        icode:ifun <- 3:0

register operands are loaded:

            rA:rB <- M_1[PC+1]
        rA:rB <-M_1[0x001]
            rA:rB <- None:%edx

constant value is loaded

        valC <- M_4[PC+2]
        valC  <-M_4[0x002]
            valC <- 9

next instruction address is computed

        valP <- PC + 6
        valP <- 0x100 + 6
        valP <- 0x106

DECODE:

        Nothing happens at the decode stage.  Why?

EXECUTE:

instruction is executed as valE

        valE <- 0 + valC
        valE <- 9

MEMORY:

        Nothing happens at memory stage.  Why?

WRITE BACK:

result of execution, valE is written to register

        R[rB] <- valE
        %edx <- 9

PC UPDATE

update program counter:

        PC <-valP
        PC <-0x106



0x00C:6123              subl           %edx, %ebx

FETCH:
instruction bytes are read, two 4-bit codes are interpreted:

        icode:ifun <- M_1[PC]
        icode:ifun <-M_1[0x00C]
        icode:ifun <- 6:1

register operands are loaded:

            rA:rB <- M_1[PC+1]
        rA:rB <-M_1[0x00D]
            rA:rB <- %edx:%ebx (2:3)
```

```
next instruction address is computed

        valP <- PC + 2
        valP <- 0x10C + 2
        valP <- 0x10E

DECODE
values are read from the register file

        valA <-R[%edx]
        valA <-9
        valB <-R[%ebx]
        valB <-21

EXECUTE
instruction is executed as valE

        valE <- valB OP valA
        valE <- 21 - 9
        valE <- 12

condition codes are set
        ZF <- 0
        SF <- 0
        0F <- 0

MEMORY

        No memory operations

WRITE BACK

        save valE to register B

        R[rB] <- valE
        R[%ebx] <-12

PC UPDATE

update program counter:

        PC <-valP
        PC <-0x10E



0x014: 404364000000             rmmovl  %esp, 100(%ebx)

FETCH:
instruction bytes are read, two 4-bit codes are interpreted

        icode:ifun <- M_1[PC]
        icode:ifun <-M_1[0x014[
        icode:ifun: 4:0

register operands are loaded:

        rA:rB <- M_1[PC+1]
        rA:rB < -M_1[0x015]
        rA:rB <- %esp:%ebx (4:3)
```

```
constant value is loaded:

        valC <- M_4[PC+2]
        valC <- M_4[0x016]
        valC <- 100

next instruction address is computed

        valP <- PC + 6
        valP <- 0x014 + 6
        valP <- 0x01A

DECODE
values are read from the register file

        valA <- R[%esp]
        valA <- 128
        valB <- R[%ebx]
        valB <- 12

EXECUTE
instruction is executed as valE

        valE <- valB+valC
        valE <- 12 + 100
        valE <- 112

MEMORY
value is written to memory

        M_4[valE] <- valA
        M_4[112 ] <- 128

WRITE BACK
there is no write back action.  Why?

PC UPDATE
update program counter:

        PC <-valP
        PC <-0x01A



0x01A A02F                      pushl           %edx

FETCH
instruction bytes are read, two 4-bit codes are interpreted

        icode:ifun <- M_1[PC]
        icode:ifun <-M_1[0x01A]
        icode:ifun: A:0

register operands are loaded:

        rA:rB <- M_1[PC+1]
        rA:rB < -M_1[0x01B]
        rA:rB <- %edx:none (2:15)

next instruction address is computed
```

```
        valP <- PC + 2
        valP <- 0x01A + 2
        valP <- 0x01C

DECODE
values are read from the register file

        valA <- R[%edx]
        valA <- 9
        valB <- R[%esp]
        valB <- 128

EXECUTE
instruction executed as valE

        valE <-valB + (-4)
        valE <-128 + (-4)
        valE <- 124

MEMORY
value is written to memory

        M[valE] <- valA
        M[124] <- 9

WRITE BACK
stack pointer is updated

        R[%esp] <-valE
        R[%esp] <- 124

PC UPDATE
update program counter

        PC <- valP
        PC <-0x01C
```

Question: why are we allowed to pushl %esp?
If we trace through the sequential execution cycle, we see that everything functions normally. In the memory stage, the instruction will store valA, the original value of the stack pointer, to memory. As we hoped and expected! How nice!

```
0x01E    7328000000     je      done

FETCH
instruction bytes are read, two 4-bit codes are interpreted

        icode:ifun <- M_1[PC]
        icode:ifun <-M_1[0x01E]
        icode:ifun: 7:3

constant value is loaded:

        valC <- M_4[PC+1]
        valC <- M_4[0x01F]
        valC <- 0x028
```

```
next instruction address is computed

        valP <- PC + 5
        valP <- 0x01E + 5
        valP <- 0x023

DECODE
Nothing happens on decode. Why?

EXECUTE

 Condition codes and branch conditions are checked
        Cnd <-- Cond(CC, ifun)
        Cnd <- Cond({0,0,0}, 3)
        Cnd <- 0

MEMORY
nothing to do here

WRITE BACK
nothing to do here

PC UPDATE

        PC <- Cnd ? valC : valP
        PC <- 0 ? 0x028 : 0x023
        PC <- 0x023 (branch not followed)
```