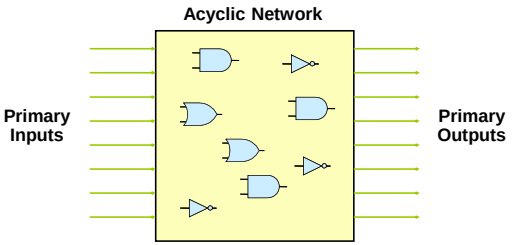
Overview of Logic Design

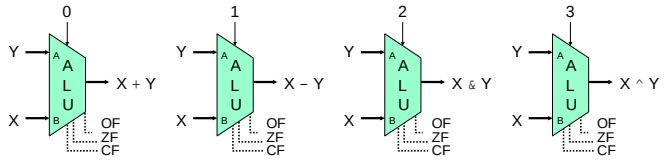
- Fundamental Hardware Requirements
  - Communication
    - How to get values from one place to another
  - Computation
  - Storage
- Bits are Our Friends
  - Everything expressed in terms of values 0 and 1
  - Communication
    - Low or high voltage on wire
  - Computation
    - Compute Boolean functions
  - Storage
    - Store bits of information

Combinational Circuits



- Acyclic Network of Logic Gates
  - Continously responds to changes on primary inputs
  - Primary outputs become (after some delay) Boolean functions of primary inputs

Arithmetic Logic Unit

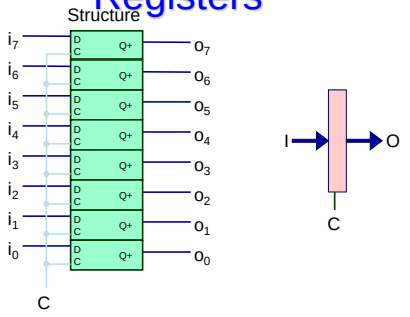


- Combinational logic
  - Continuously responding to inputs
- Control signal selects function computed
  - Corresponding to 4 arithmetic/logical operations in Y86
- Also computes values for condition codes

Sequential Logic: Memory and Control

- Sequential:
  - Output depends on the ***current*** input values and the ***previous*** sequence of input values.
  - Are Cyclic:
    - Output of a gate feeds its input at some future time.
  - Memory:
    - Remember results of previous operations
    - Use them as inputs.
  - Example of use:
    - Build registers and memory units.

Registers

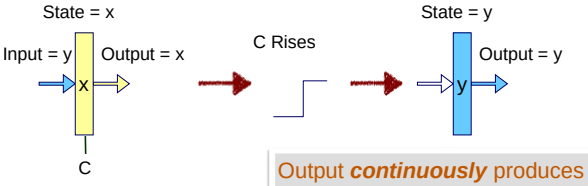


- Stores several bits of data
- Collection of edge-triggered latches (D Flip-flops)
- Loads input on rising edge of the C signal

34

34

Register Operation



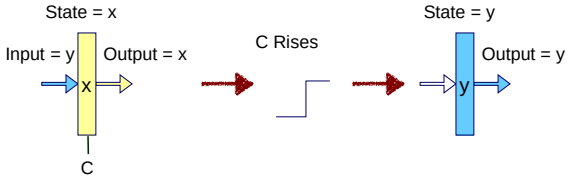
Output ***continuously*** produces y after the rising edge unless you cut off power.

- Stores data bits
- For most of time acts as barrier between input and output
- As C rises, loads input
- So you'd better compute the input before the C signal rises if you want to store the input data to the register

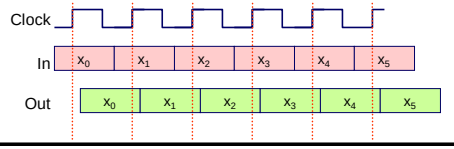
35

35

Clock Signal



- A special C: periodically oscillating between 0 and 1
- That's called the **clock** signal. Generated by a crystal oscillator inside your computer

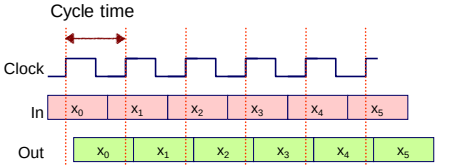


36

36

Clock Signal

- Cycle time of a clock signal: the time duration between two rising edges.
- Frequency of a clock signal: how many rising (falling) edges in 1 second.
- 1 GHz CPU means the clock frequency is 1 GHz
  - The cycle time is 1/10^9 = 1 ns

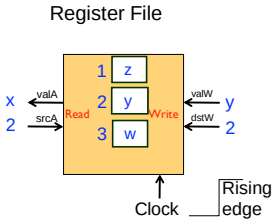


37

37

Register File

- A register file consists of a set of registers that you can individual read from and write to.
- To read: give a register file ID, and read the stored value out
- To write: give a register file ID, a new value, overwrite the old value
- How do we build a register file out of individual registers??

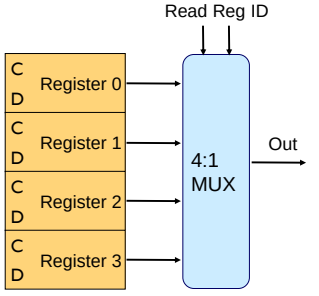


38

38

Register File Read

- Continuously read a register independent of the clock signal

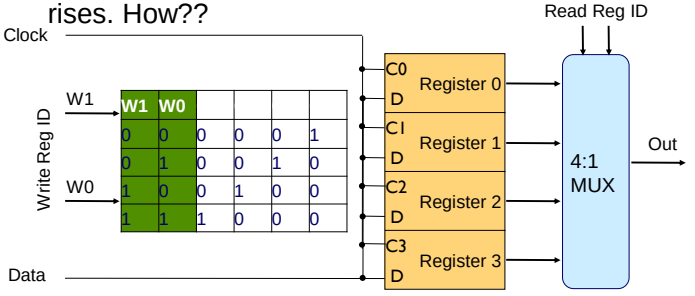


39

39

Register File Write

- Only write the a specific register when the clock rises. How??



40

40

Decoder

W1	W0				
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

W0
W1

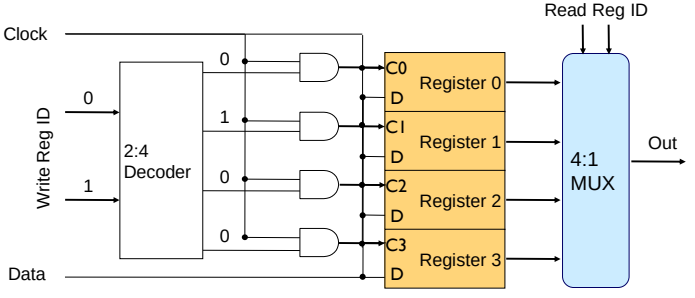
_C0
_C1
_C2
_C3

C0 = !W1 & !W0
C1= !W1 & W0
C2 = W1 & !W0
C3 = W1 & W0

41

41

Register File Write



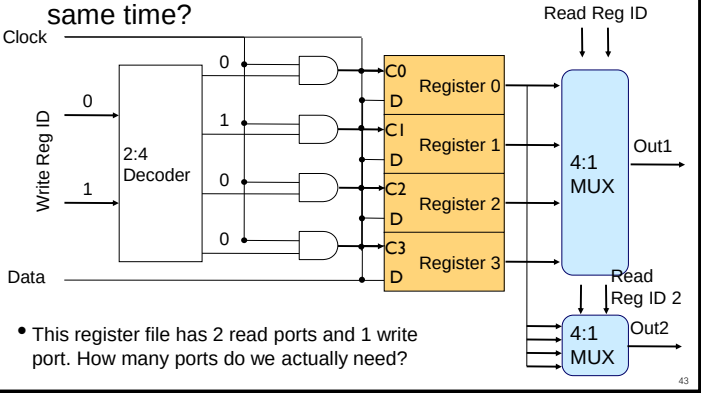
- This implementation can read 1 register and write 1 register at the same time: 1 read port and 1

42

42

Multi-Port Register File

- What if we want to read multiple registers at the same time?



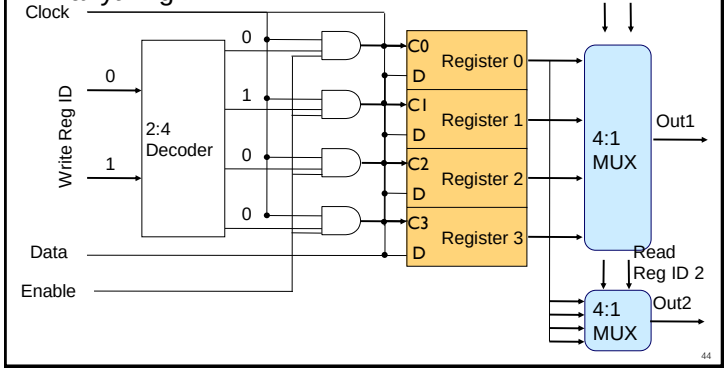
- This register file has 2 read ports and 1 write port. How many ports do we actually need?

43

43

Multi-Port Register File

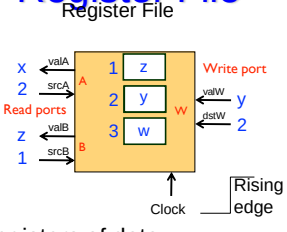
- Is this correct? What if we don't want to write anything?



44

44

Register File



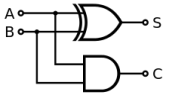
- Stores multiple registers of data
  - Address input specifies which register to read or write
- Register file is a form of Random-Access Memory (RAM)
- Multiple Ports: Can read and/or write multiple words in one cycle. Each port has separate address and data input/output

45

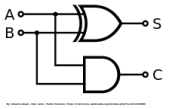
45

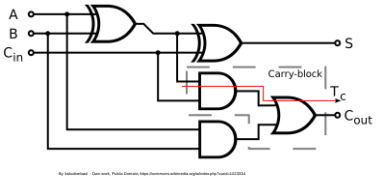
Breakout

- What does this circuit compute? How many 2-input NAND gates will you need?



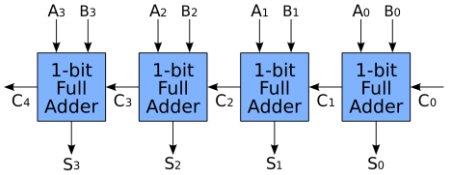
Half and Full Adders





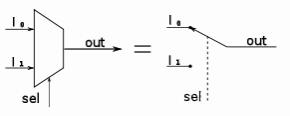
Ripple Carry and Lookahead Adders

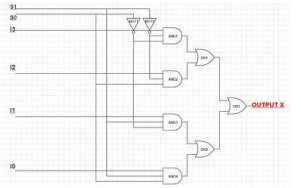
- Ripple Carry Adder
  - Time?



- Carry Lookahead Adder
  - Generate carries in parallel
  - Logarithmic versus linear time

Multiplexer

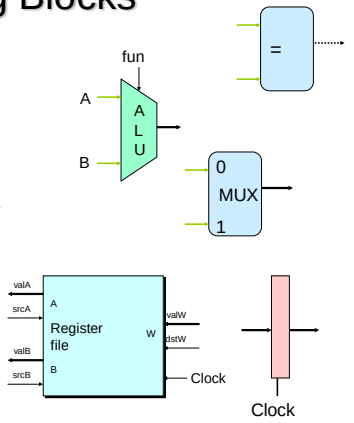




Aside: The number of inputs of a gate (fan-in) and the number of outputs of a gate (fan-out) will affect the gate delay

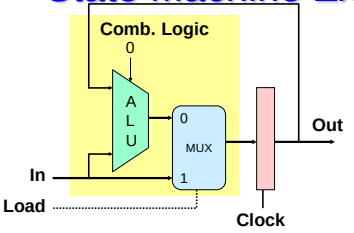
Building Blocks

- Combinational Logic
  - Compute Boolean functions of inputs
  - Continuously respond to input changes
  - Operate on data and implement control
- Storage Elements
  - Store bits
  - Addressable memories
  - Non-addressable registers
  - Loaded only as clock rises

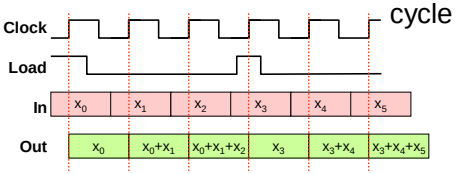


50

State Machine Example



- Accumulator circuit
- Load or accumulate on each cycle



51

Hardware Components of a Computer System

- Processor
  - Datapath
  - Control
- Memory
- Input and Output devices

52

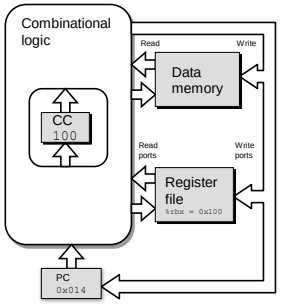
Sequential Architecture: Microarchitecture Overview

Think of it as a state machine

Every cycle, one instruction gets executed. At the end of the cycle, architecture states get modified.

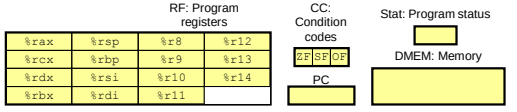
States (All updated as clock rises)

- PC register
- Cond. Code register
- Data memory
- Register file



53

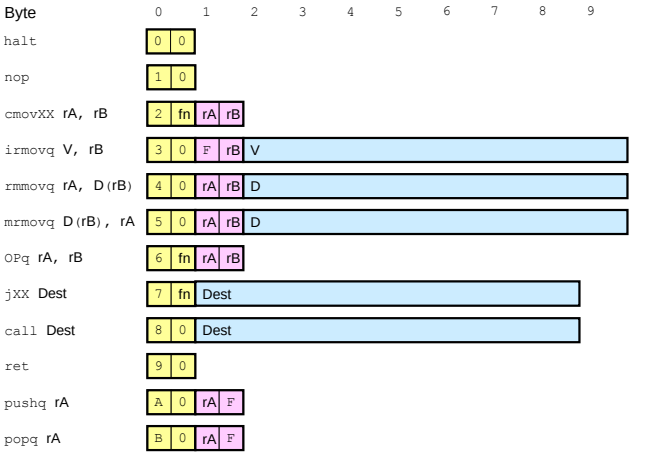
Y86-64 Processor State



- Program Registers
  - 15 registers (omit `%r15`). Each 64 bits
- Condition Codes
  - Single-bit flags set by arithmetic or logical instructions
    - ZF: Zero SF:Negative OF: Overflow
- Program Counter
  - Indicates address of next instruction
- Program Status
  - Indicates either normal operation or some error condition
- Memory
  - Byte-addressable storage array
  - Words stored in little-endian byte order

54

Y86-64 Instruction Set #1



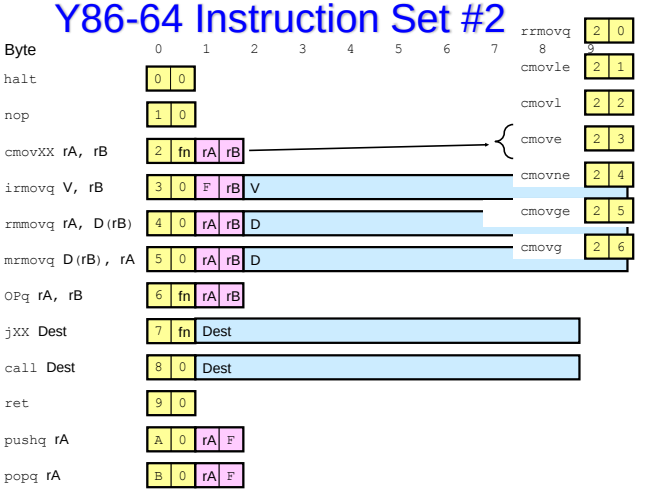
55

Y86-64 Instructions

- Format
  - 1–10 bytes of information read from memory
    - Can determine instruction length from first byte
    - Not as many instruction types, and simpler encoding than with x86-64
  - Each accesses and modifies some part(s) of the program state

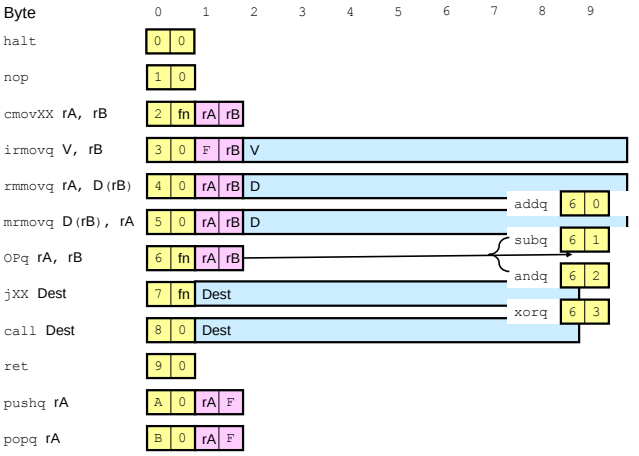
56

Y86-64 Instruction Set #2



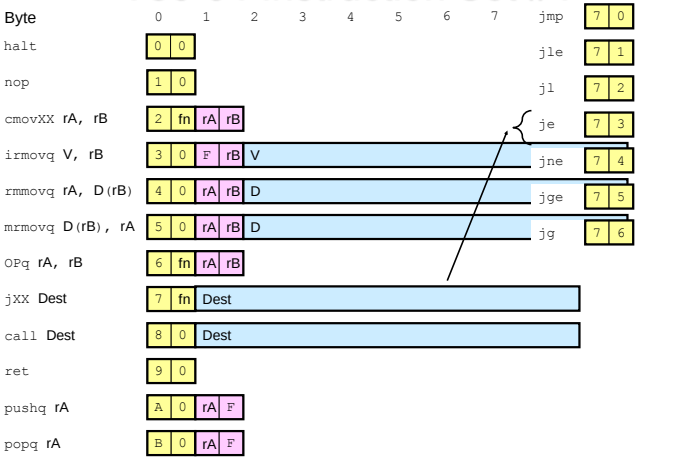
57

Y86-64 Instruction Set #3



58

Y86-64 Instruction Set #4



59

Encoding Registers

- Each register has 4-bit ID

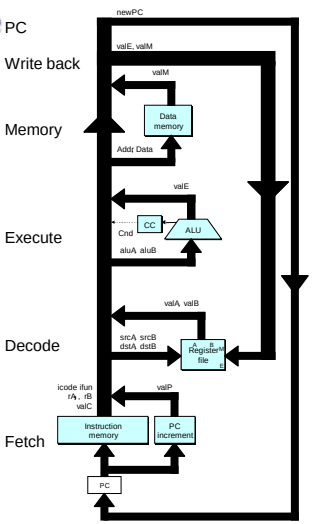
%rax	0	%r8	8
%rcx	1	%r9	9
%rdx	2	%r10	A
%rbx	3	%r11	B
%rsp	4	%r12	C
%rbp	5	%r13	D
%rsi	6	%r14	E
%rdi	7	No Register	F

  - Same encoding as in x86-64
- Register ID 15 (`0xF`) indicates "no register"
  - Will use this in our hardware design in multiple places

60

SEQ Hardware Structure

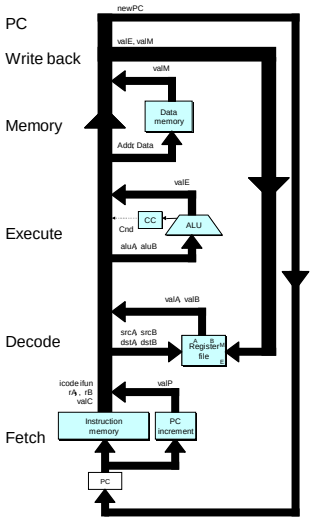
- State
  - Program counter register (PC)
  - Condition code register (CC)
  - Register File
  - Memories
    - Access same memory space
    - Data: for reading/writing program data
    - Instruction: for reading instructions
- Instruction Flow
  - Read instruction at address specified by PC
  - Process through stages
  - Update program counter



61

SEQ Stages

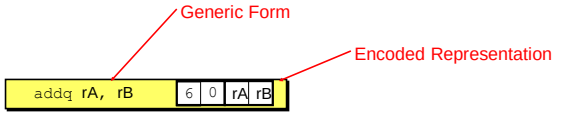
- Fetch
  - Read instruction from instruction memory
- Decode
  - Read program registers
- Execute
  - Compute value or address
- Memory
  - Read or write data
- Write Back
  - Write program registers
- PC
  - Update program counter



62

Instruction Example

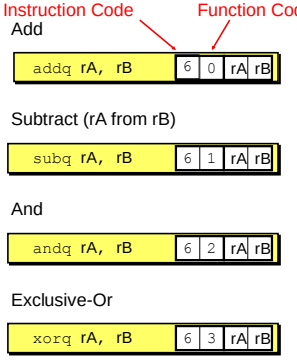
- Addition Instruction



  - Add value in register rA to that in register rB
    - Store result in register rB
    - Note that Y86-64 only allows addition to be applied to register data
  - Set condition codes based on result
  - e.g., `addq %rax,%rsi` Encoding: `60 06`
  - Two-byte encoding
    - First indicates instruction type
    - Second gives source and destination registers

63

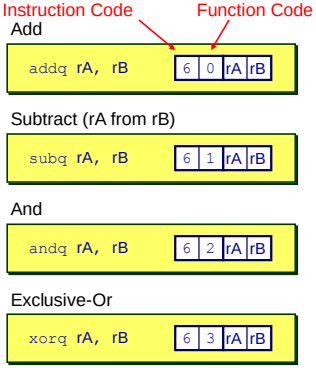
Arithmetic and Logical Operations



- Refer to generically as "OPq"
- Encodings differ only by "function code"
  - Low-order 4 bytes in first instruction word
- Set condition codes as side effect

64

Arithmetic and Logical Operations



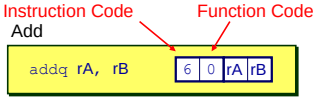
- Refer to generically as "OPq"
- Encodings differ only by "function code"
  - Low-order 4 bits in first instruction word
- Set condition codes as side effect

65

65

Executing an ADD instruction

– How does the processor execute `addq %rax,%rsi`
– The binary encoding is `60 06`

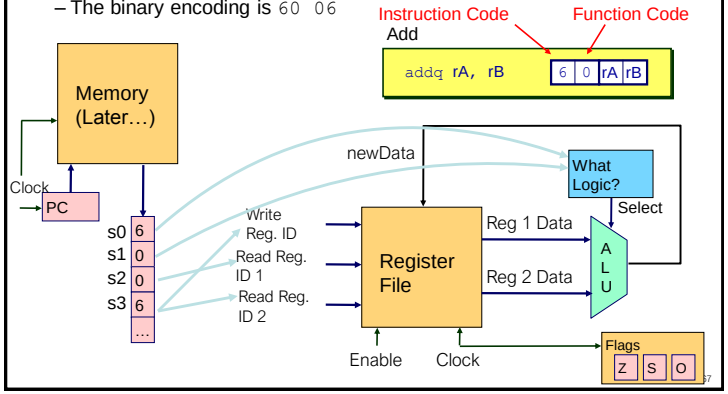


66

66

Executing an ADD instruction

– How does the processor execute `addq %rax,%rsi`
– The binary encoding is `60 06`

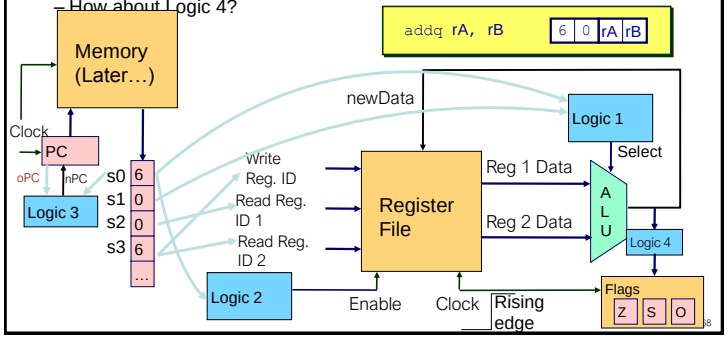


67

Executing an ADD instruction

– Logic 1: if (s0 == 6) select = s1;
– Logic 2: if (s0 == 6) Enable = 1; else Enable = 0;
– Logic 3: if (s0 == 6) nPC = oPC + 2;
– How about Logic 4?

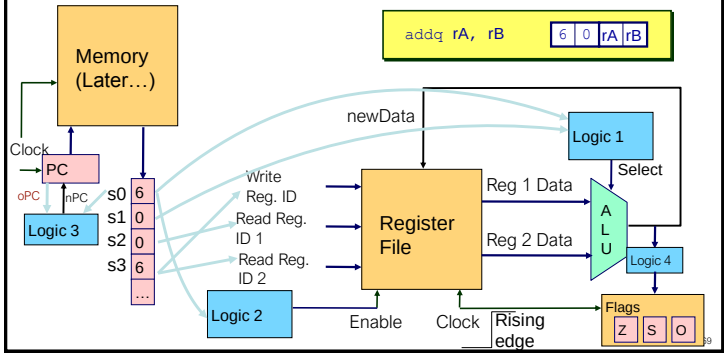
How do these logics get implemented?



68

Executing an ADD instruction

– When the rising edge of the clock arrives, the RF/PC/Flags will be written.
– So the following has to be ready: newData, nPC, which means Logic1, Logic2, Logic3, and Logic4 has to finish.

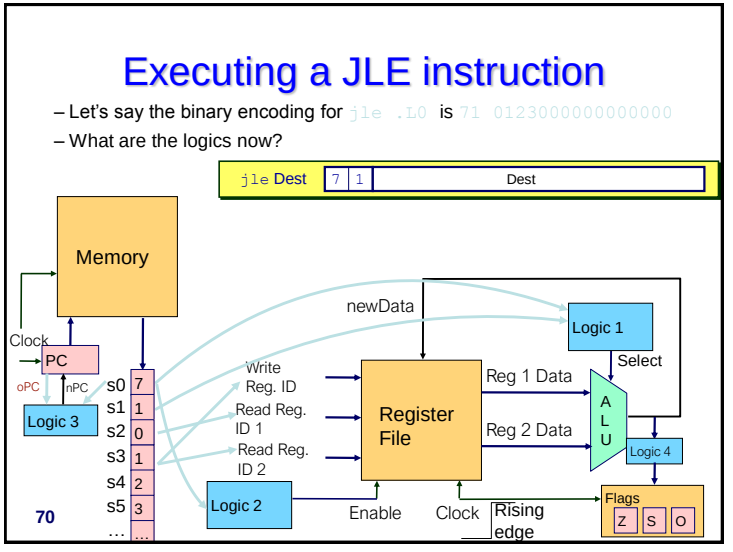


69

Executing a JLE instruction

– Let's say the binary encoding for `jle .L0` is `71 0123000000000000`

– What are the logics now?

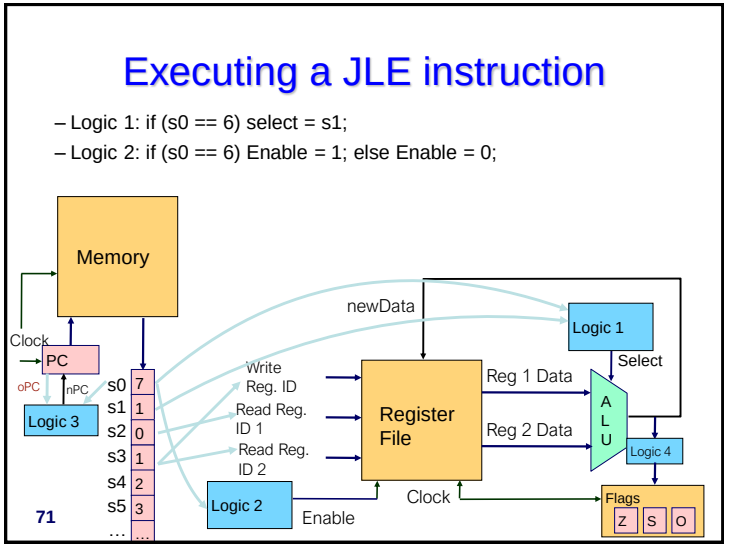


70

Executing a JLE instruction

– Logic 1: if (s0 == 6) select = s1;

– Logic 2: if (s0 == 6) Enable = 1; else Enable = 0;



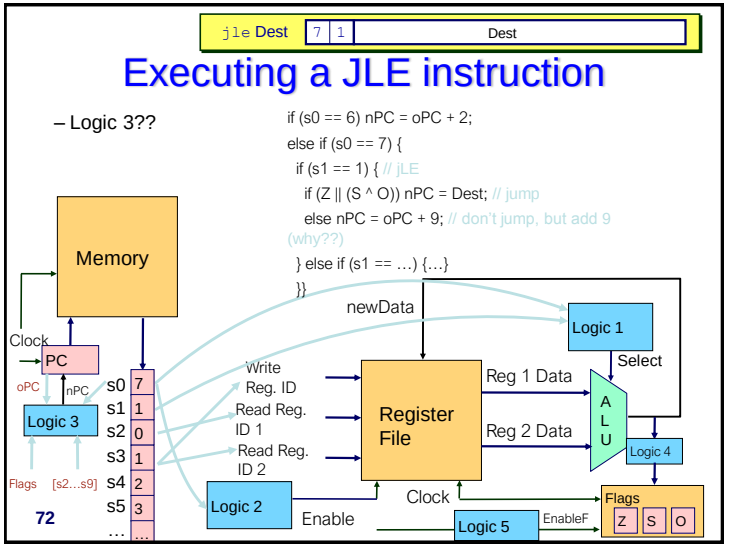
71

Executing a JLE instruction

`jle Dest` | `7` | `1` | Dest

– Logic 3??

```
if (s0 == 6) nPC = oPC + 2;
else if (s0 == 7) {
  if (s1 == 1) { // jLE
    if (Z || (S ^ O)) nPC = Dest; // jump
    else nPC = oPC + 9; // don't jump, but add 9 (why??)
  } else if (s1 == …) {…}
  }}
```



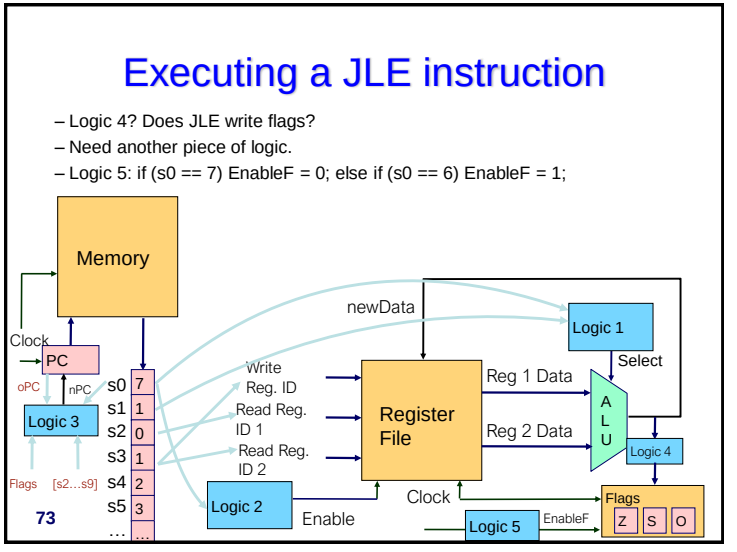
72

Executing a JLE instruction

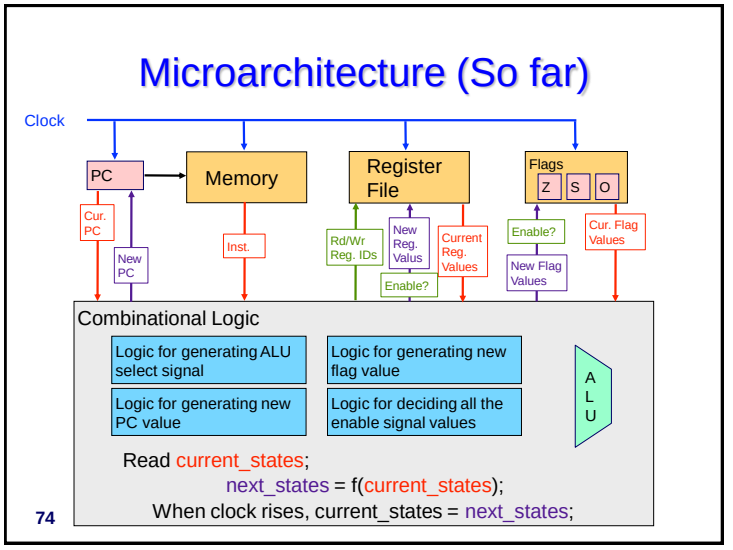
– Logic 4? Does JLE write flags?

– Need another piece of logic.

– Logic 5: if (s0 == 7) EnableF = 0; else if (s0 == 6) EnableF = 1;



73


Microarchitecture (So far)
Clock
PC
Memory
Register File
Flags
Z
S
O
Cur. PC
New PC
Inst.
Rd/Wr Reg. IDs
New Reg. Valus
Enable?
Current Reg. Values
Enable?
New Flag Values
Cur. Flag Values
Combinational Logic
Logic for generating ALU select signal
Logic for generating new flag value
Logic for generating new PC value
Logic for deciding all the enable signal values
A L U
Read current_states;
next_states = f(current_states);
When clock rises, current_states = next_states;
74

74