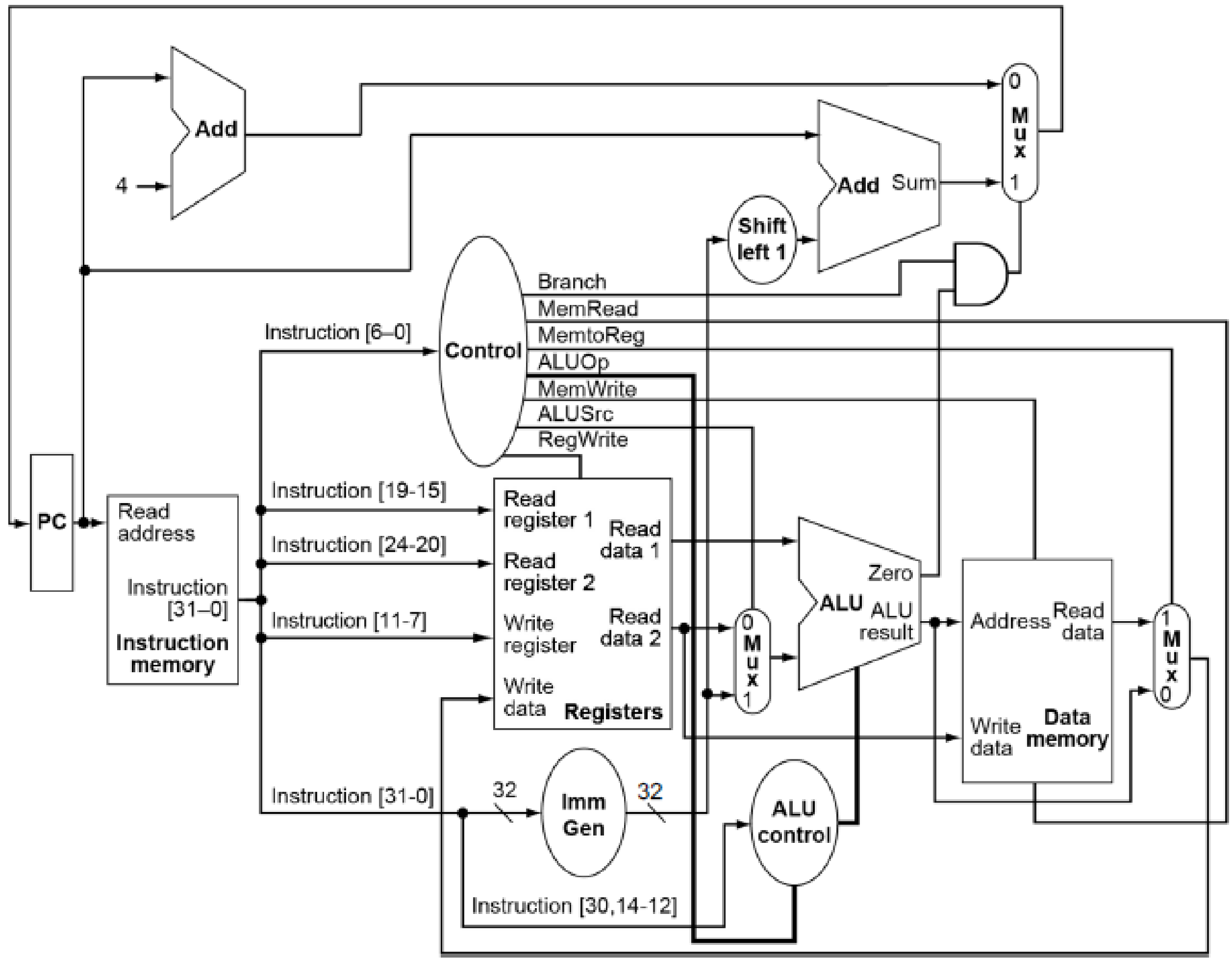
Single Cycle Processor & Pipelined Processor (T5-T6)

VE370FA22 TA Jiajun Gu

Single Cycle Processor



- Only **one instruction** can be executed in a clock cycle
- ***Only PC here is a register! (Only PC and register file are connected to clock)***

ALU Control (Unit)

ALU control Signal	Function
0000	AND
0001	OR
0010	add
0110	subtract

Input:

- ALUOp: 2-bit derived from opcode
- Instruction[30, 14-12]

Output: 4-bit ALU control signal

Instr.	Operation	ALU function	Opcode field	ALUOp (input)	i[30] (input)	funct3 (input)	ALU control (output)
lw	load register	add	XXXXXXX	00	X	010	0010
sw	store register	add	XXXXXXX	00	X	010	0010
beq	branch on equal	subtract	XXXXXXX	01	X	000	0110
R-type	add	add	100000	10	0	000	0010
	subtract	subtract	100010		1	000	0110
	AND	AND	100100		0	111	0000
	OR	OR	100101		0	110	0001

Additional bit (instruction[30]) is needed for R-type

Type	Field					
	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits
R-type	funct7	rs2	rs1	funct3	rd	opcode

- `add` (funct7): 0000000
- `sub` (funct7): 0100000

Control Signals

Make sure you are clear about everything in this table!

Inst.	ALU Src	Reg Dst	ALU Op	Mem Write	Mem Read	Branch	Memto Reg	Reg Write
add	0		10	0	0	0	0	1
addi	1		?	0	0	0	0	1
lw	1		00	0	1	0	1	1
sw	1		00	1	0	0	X	0
beq	0		01	0	0	1	X	0

Performance Evaluation

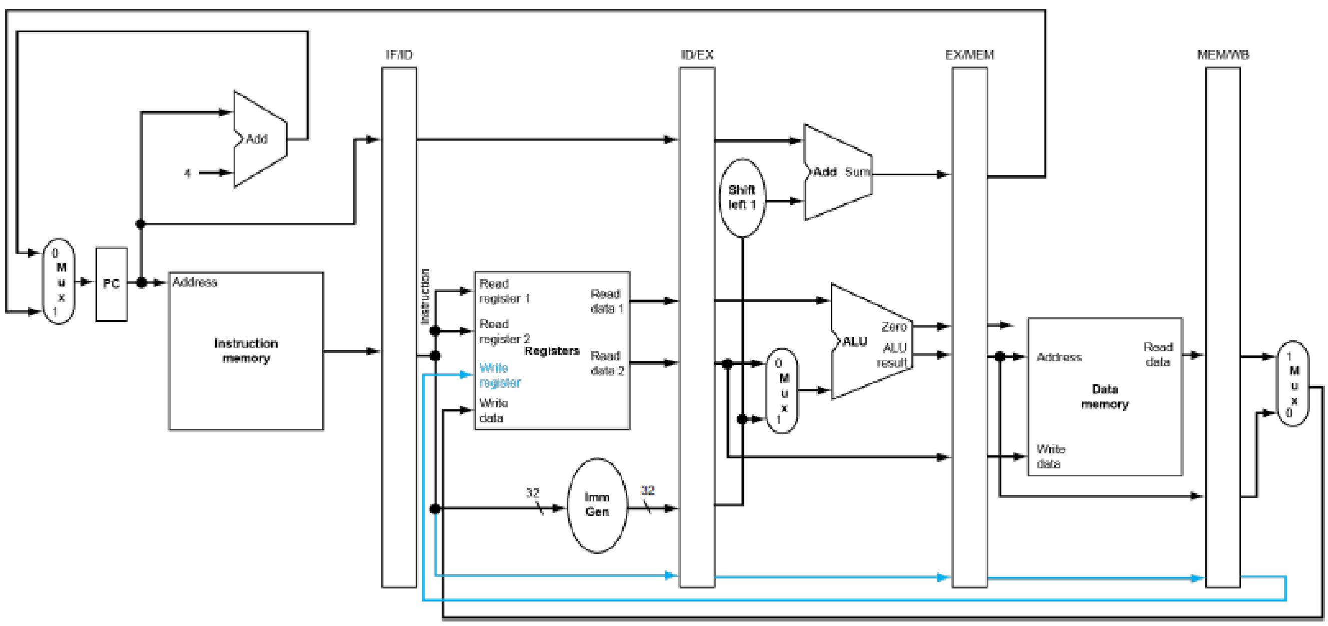
- CPU (Execution) Time = CPU Clock Cycles / Clock Rate
- Clock Cycles = Instruction Count (IC) $\times$ Cycles Per Instruction (CPI)
- Critical path of the combination logic determines the clock cycle time
  - Critical path: the path of the instruction that takes the **longest** time to finish

Pipeline Processor

Motivation: much time wasted on waiting for a single cycle processor

- Clock cycle time is fixed
- The most time-consuming instruction determines the clock cycle

General view (not complete):



5 Stages

Use registers to hold information from previous cycle between stages.

- IF: Instruction fetch from memory
- ID: Instruction decode & register read
- EX: Execute operation or calculated address
- MEM: Access memory operand
- WB: Write result back to register

Homework 1

Ex. 6

6. (5 points) Assume x5 holds the value 0x11010000. What is the value of x6 after the following instructions?

```
        addi x6, x0, 1
        bge x5, x0, ELSE
        jal x0, DONE
ELSE:   ori x6, x0, 2
DONE:   lui x6, 0xFFFFF
```

Ex. 8

8. (20 points) Translate function `f` into RISC-V assembly language following function calling conventions. Assume the function declaration for `g` is `int g(int a, int b)`. The code for function f is as follows:

```
int f(int a, int b, int c, int d){
    return g(g(a,c), b-d);
}
```

- Convention:
  - x10-x11: function arguments/results
  - x12-x17: function arguments
- Non-leaf funtion --> Save x1 into stack
- Save contents that you will used (but may be modified by other functions) into stack!
- Always remember to restore contents saved in the stack and the stack pointer
  - e.g., `sub x5, x11, x13` (x5 = b - d) before calling `g`

Ex. 9

9. (10 points) Right before your function f from Problem 8 returns, what do we know about contents of registers `x10-x14`, `x8`, `x1`, and `sp`? Keep in mind that we know what the entire function `f` looks like, but for function `g` we only know its declaration.

- x10-x14:
    - If you save them, they will be the same as their value right after entering the function.
    - If you don't save them, they may be changed by the callee (`g` in this case), so we have no idea about their contents.
- x8: Frame pointer (fp). Its use is optional and actually not used in `f`, so we don't know its content.
- x1: We don't know the exact content in x1, but we do know it must be equal to the return address set by instruction `jal x1, f` that calls `f`.
- sp: We don't know the precise content of sp, but we do know it's identical to what it is when `f` was called.