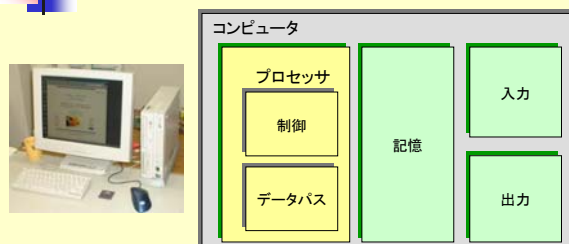


## 計算機アーキテクチャ 第一 (E)

### 5. プロセッサに関する議論

吉瀬 謙二 計算工学専攻  
kise\_at\_cs.titech.ac.jp  
W641講義室 木曜日13:20 - 14:50

## コンピュータ(ハードウェア)の古典的な要素



プロセッサは記憶装置から命令とデータを取り出す。入力装置はデータを記憶装置に書き込む。出力装置は記憶装置からデータを読み出す。制御装置は、データバス、記憶装置、入力装置、そして出力装置の動作を指定する信号を送る。

出典: パターソン & ヘネシー、コンピュータの構成と設計

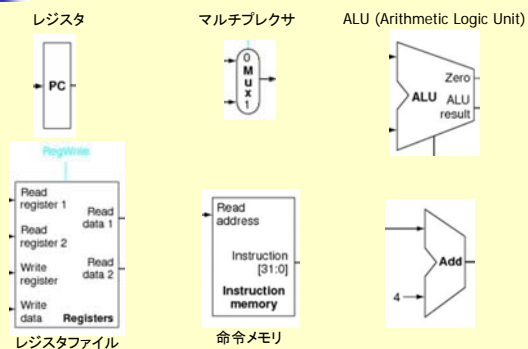
2

## MIPSの基本的な5つのステップ(ステージ)

- **IFステージ**  
メモリから命令をフェッチする。
- **IDステージ**  
命令をデコード(解読)しながら、レジスタの値を読み出す。
- **EXステージ**  
命令操作の実行またはアドレスの生成を行う。
- **MEMステージ**  
必要であれば、データ・メモリ中のオペランドにアクセスする。
- **WBステージ**  
必要であれば、結果をレジスタに書き込む。

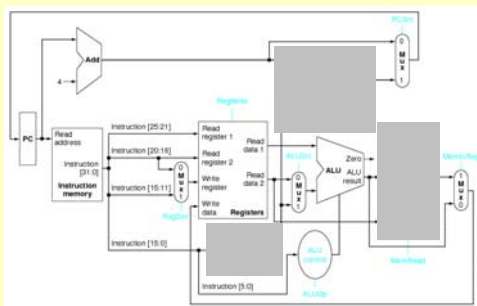
3

## 主な構成要素(1)



## プロセッサのデータパス(シングル・サイクル)

Op rs rt rd shamt funct  
0x800 add \$t0, \$s1, \$s2 [ add \$8, \$17, \$18 ]



## Machine Language - Add Instruction

- Instructions, like registers and words of data, are **32 bits long**
- Arithmetic Instruction Format (R format):  

```

      add $t0, $s1, $s2
    
```

op	rs	rt	rd	shamt	funct
6-bits	5-bits	5-bits	5-bits	5-bits	6-bits

  - op 6-bits opcode that specifies the operation
  - rs 5-bits register file address of the first source operand
  - rt 5-bits register file address of the second source operand
  - rd 5-bits register file address of the result's destination
  - shamt 5-bits shift amount (for shift instructions)
  - funct 6-bits function code augmenting the opcode

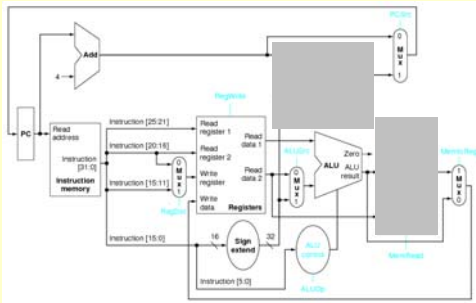
Adapted from Computer Organization and Design, Patterson & Hennessy, © 2005

6

### プロセッサのデータパス(シングル・サイクル)

op	rs	rt	16 bit immediate
----	----	----	------------------

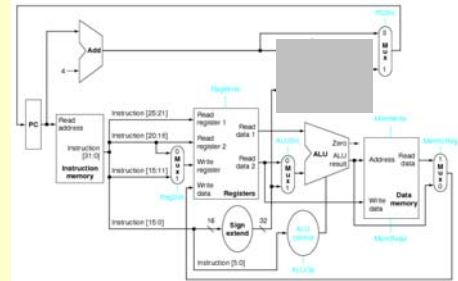
 I format  
 0x804 addi \$t0, \$t1, -1 [ addi \$8, \$9, -1 ]



### プロセッサのデータパス(シングル・サイクル)

op	rs	rt	16 bit immediate
----	----	----	------------------

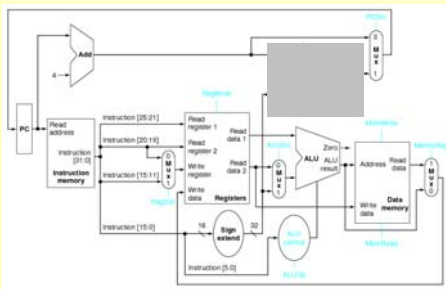
 I format  
 0x808 lw \$t0, 24(\$s2) [ lw \$8, 24(\$18) ]



### プロセッサのデータパス(シングル・サイクル)

op	rs	rt	16 bit immediate
----	----	----	------------------

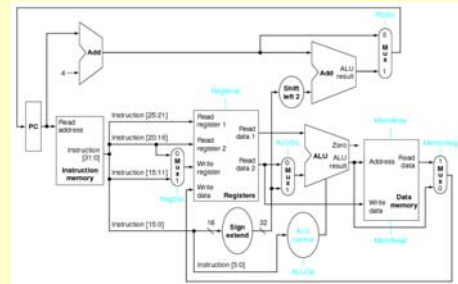
 I format  
 sw \$t0, 24(\$s2) [ sw \$8, 24(\$18) ]



### プロセッサのデータパス(シングル・サイクル)

op	rs	rt	16 bit immediate
----	----	----	------------------

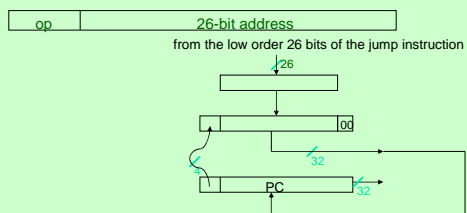
 I format  
 beq \$s0, \$s1, Label [ beq \$16, \$17, Label ]



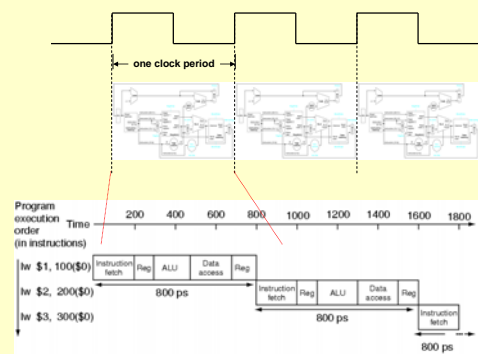
### Other Control Flow Instructions

- MIPS also has an **unconditional branch** instruction or **jump** instruction:  
 j label #go to label

- Instruction Format (J Format):



### プロセッサのデータパス(シングル・サイクル)



# エッジトリガ方式による設計

The diagram illustrates an edge-triggered design. It consists of three main components connected in a loop: State Element 1, Combinational logic, and State Element 2. A clock cycle waveform is shown below the components, indicating the timing of the data flow.


The detailed logic diagram shows a sequential circuit with feedback. It includes combinational logic blocks (AND, OR, NOT, MUX), state elements (D flip-flops), and a clock signal. The circuit is designed to implement a specific function, with the clock signal triggering the state transitions.

## プロセッサのデータパス(マルチ・サイクル)

## プロセッサのデータパス(マルチ・サイクル)

Diagram illustrating instruction pipelining with three instructions (lw \$1, 100(\$0), lw \$2, 200(\$0), lw \$3, 300(\$0)) and their execution stages (Instruction fetch, Reg, ALU, Data access, Reg) over time (0 to 1800 ps). The diagram shows the progression of instructions through the pipeline stages, with a 200 ps delay between instructions.

## パイプライン処理 (pipelining)



The diagram illustrates the concept of pipelining in a car production line. It shows a sequence of stages where cars move through the process. The stages are labeled as follows:

- 1. 塗装ライン (Painting Line)
- 2. トリムライン (Trim Line)
- 3. インストルメントパネル (Instrument Panel)
- 4. 電気配線 (Electrical Wiring)
- 5. フォイナルライン (Final Line)
- 6. 車検待機ライン (Waiting Line for Inspection)
- 7. 車検通過ライン (Passed Inspection Line)
- 8. 車検通過後 (After Inspection)

The diagram shows how cars move through these stages, with some cars being in multiple stages simultaneously, demonstrating the concept of pipelining.

**Sequential Execution:**

- Instruction 1 (lw \$1, 100(\$0)): 800 ps (IF, ID, EX, WB)
- Instruction 2 (lw \$2, 200(\$0)): 800 ps (IF, ID, EX, WB)
- Instruction 3 (lw \$3, 300(\$0)): 800 ps (IF, ID, EX, WB)
- Total Time: 1800 ps

**Pipelined Execution:**

- Instruction 1 (lw \$1, 100(\$0)): 200 ps (IF, ID, EX, WB)
- Instruction 2 (lw \$2, 200(\$0)): 200 ps (IF, ID, EX, WB)
- Instruction 3 (lw \$3, 300(\$0)): 200 ps (IF, ID, EX, WB)
- Total Time: 1400 ps

**Speedup:**  $\frac{1800 \text{ ps}}{1400 \text{ ps}} = 1.25 \times$

## プロセッサの3つの実現方式

- シングル・サイクル
- マルチ・サイクル
- パイプライン処理

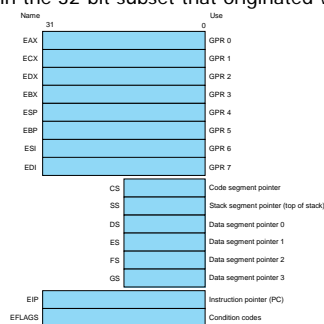
19

## Discussion

- RISC (Reduced Instruction Set Computer)
- CISC (Complex Instruction Set Computer)

## IA-32 Registers and Data Addressing

- Registers in the 32-bit subset that originated with 80386



21

## IA-32 Typical Instructions

- Four major types of integer instructions:
  - Data movement including move, push, pop
  - Arithmetic and logical (destination register or memory)
  - Control flow (use of condition codes / flags)
  - String instructions, including string move and string compare

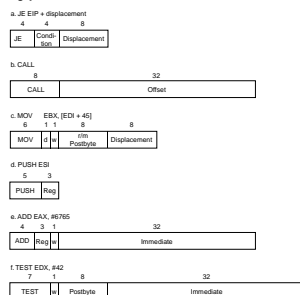
Instruction	Function
JE name	If equal (condition code) (EIP=name); EIP-128 ≤ name < EIP+128
JMP name	EIP=name
CALL name	SP=SP-4; M[SP]=EIP+5; EIP=name;
MOV EBX, [EDI+45]	EBX=M[EDI+45]
PUSH ESI	SP=SP-4; M[SP]=ESI
POP EDI	EDI=M[SP]; SP=SP+4
ADD EAX, #6765	EAX=EAX+6765
TEST EDX, #42	Set condition code (flags) with EDX and 42
MOVSX	M[EDI]=M[ESI]; EDI=EDI+4; ESI=ESI+4

FIGURE 2.43 Some typical IA-32 instructions and their functions. A list of frequent operations appears in Figure 2.44. The CALL saves the EIP of the next instruction on the stack. (EIP is the Intel PC.)

22

## IA-32 instruction Formats

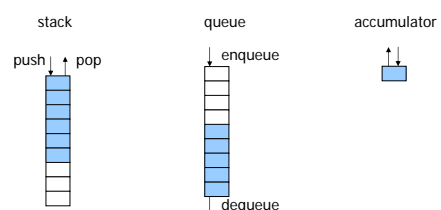
- Typical formats: (notice the different lengths)



23

## 基本記憶方式

- general-purpose register architecture
- stack architecture
- queue architecture
- accumulator architecture





## オペランド数

- 3オペランド
- 2オペランド
  - SuperH ADD Rm, Rn :  $Rn \leftarrow Rn + Rm$

- MIPS Arithmetic Instruction Format (R format):

