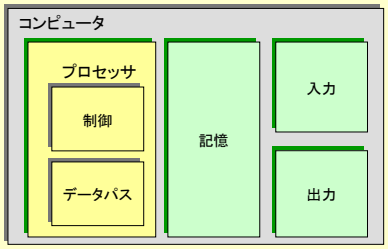
2010-05-13

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 ー 14:50

コンピュータ(ハードウェア)の古典的な要素



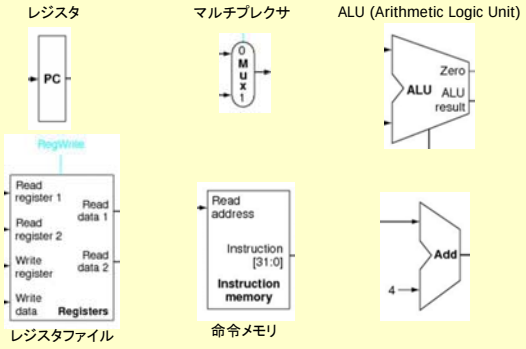
プロセッサは記憶装置から命令とデータを取り出す。入力装置はデータを記憶装置に書き込む。出力装置は記憶装置からデータを読みだす。制御装置は、データパス、記憶装置、入力装置、そして出力装置の動作を指定する信号を送る。

出典：パターソン & ヘネシー、コンピュータの構成と設計

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.

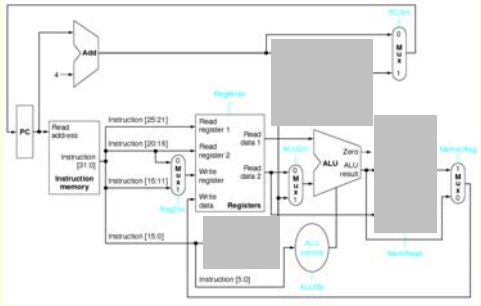
主な構成要素(1)



プロセッサの**データパス**(シングル・サイクル)

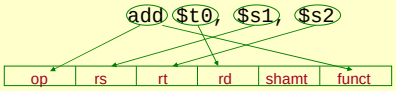
op	rs	rt	rd	shamt	funct

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):



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

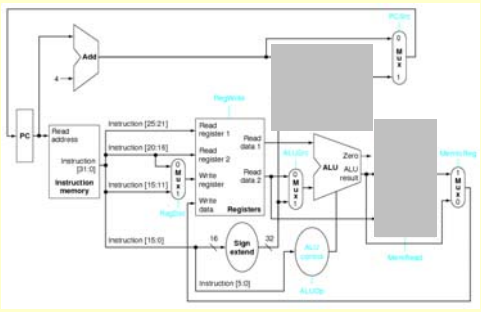
Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

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]

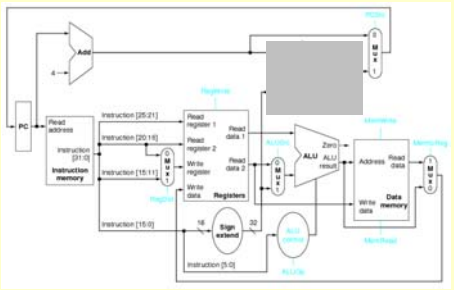


プロセッサのデータパス（シングル・サイクル）

op	rs	rt	16 bit immediate

I format

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

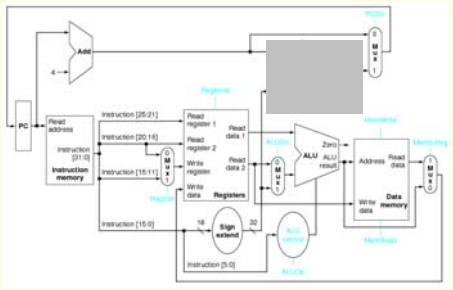


プロセッサのデータパス（シングル・サイクル）

op	rs	rt	16 bit immediate

I format

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

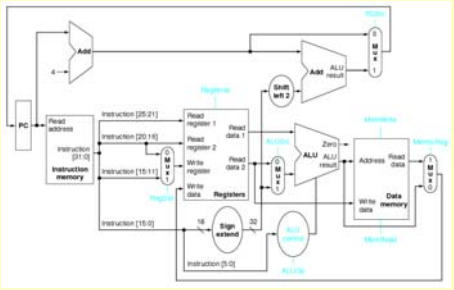


プロセッサのデータパス（シングル・サイクル）

op	rs	rt	16 bit immediate

I format

beq $s0, $s1, Label [beq $16, $17, Label]



Other Control Flow Instructions

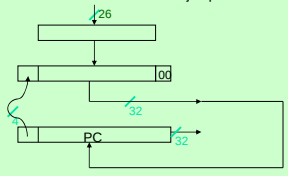
- MIPS also has an **unconditional branch** instruction or jump instruction:

```
j   label       #go to label
```

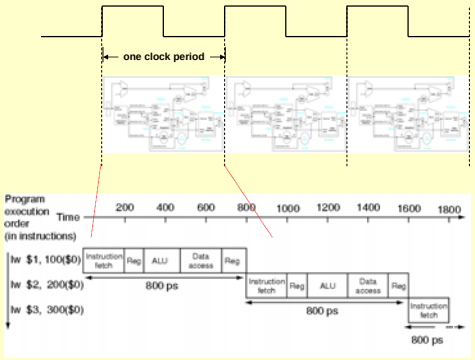
- Instruction Format (J Format):

op	26-bit address

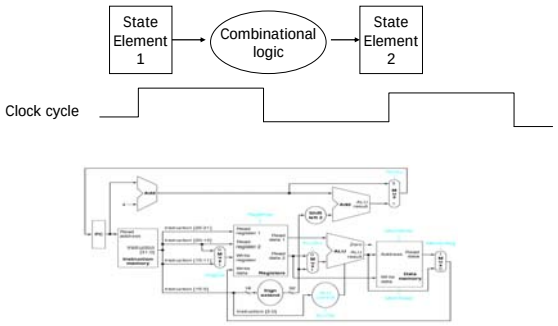
from the low order 26 bits of the jump instruction



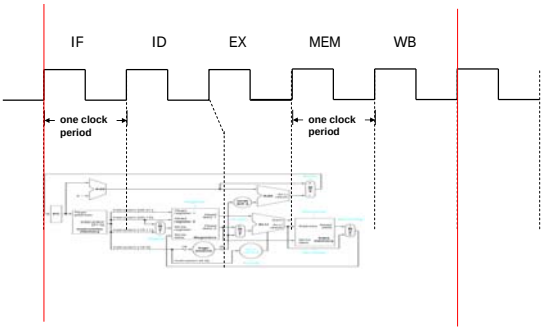
プロセッサのデータパス（シングル・サイクル）



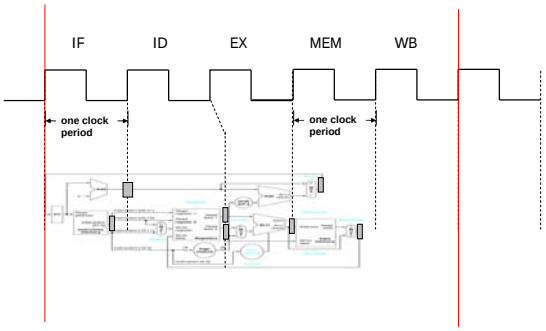
エッジトリガ方式による設計



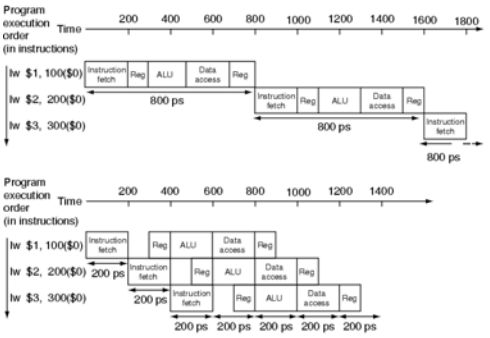
プロセッサのデータパス（マルチ・サイクル）



プロセッサのデータパス（マルチ・サイクル）

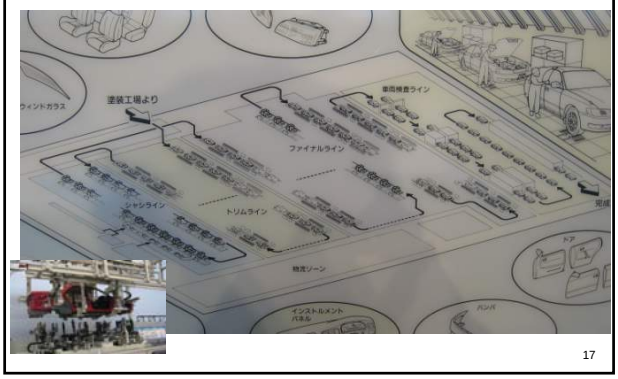


パイプライン処理 (pipelining)



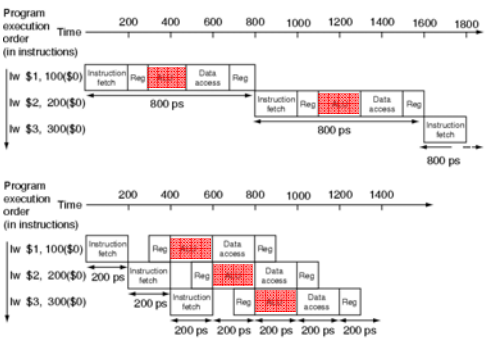
16

パイプライン処理 (pipelining)



17

パイプライン処理 (pipelining)



18

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

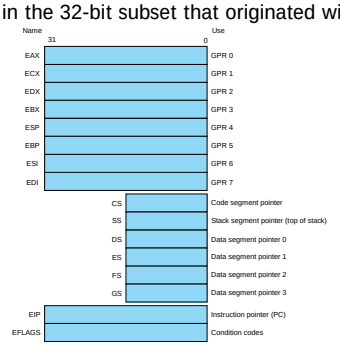
19

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

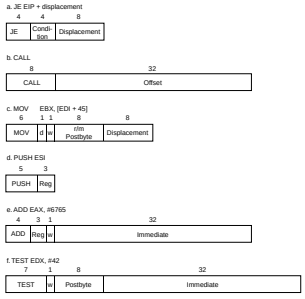
Instruction	Function
JE name	if equal(condition code) {EIP=name}; EIP-128 ≤ name < EIP+128
JMP name	EIP=name
CALL name	SP=SP-4; M[SP]=EIP+5; EIP=name;
MOVW EBX,[EDI+45]	EBX=M[EDI+45]
PUSH ESI	SP=SP-4; M[SP]=ESI
POP EDI	EDI=M[SP]; SP=SP+4
ADD EAX,#6765	EAX= EAX+6765
TEST EDX,#42	Set condition code (flags)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

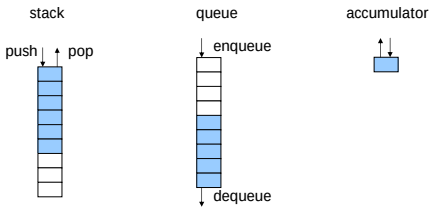
- Typical formats: (notice the different lengths)



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 <- Rn + Rm

- MIPS Arithmetic Instruction Format (R format):