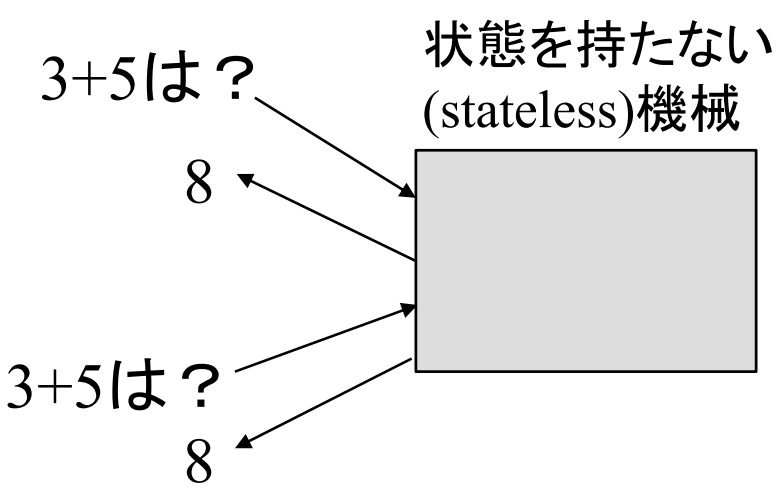
第6回
2018/12/21
順序回路と最初のプロセッサ設計

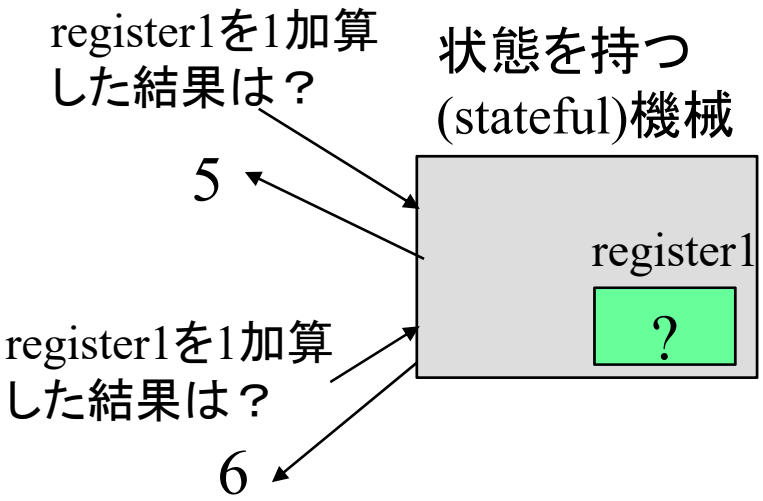
図の多くはPatterson, Hennessy: Computer organization and design 5th editionより引用

前回の復習と順序回路

- 加算、減算、乗算などを行うArithmetic Logic Unit (ALU)はどのような回路で構成されるか
  - 組み合わせ回路と呼ばれる回路から成る
  - 「状態」を持たず、(数学の)関数的に動作
- しかし、これだけではプロセッサにならない!
  - レジスタ・メモリなど、状態の表現が必要 → 順序回路



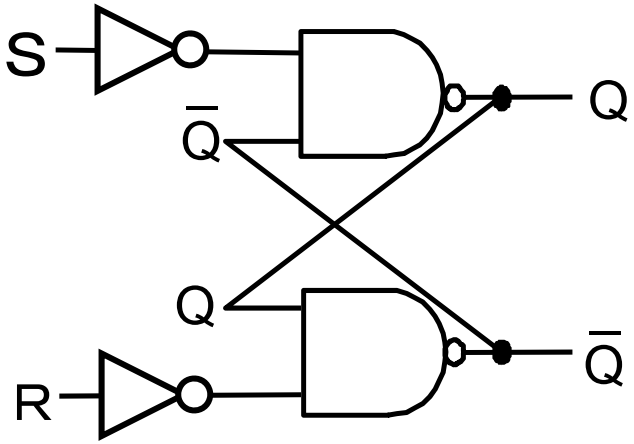
同じ入力には同じ出力



同じ入力で出力が違いうる

最も簡単な順序回路：フリップフロップ

- フリップフロップ (Filp-Flop)：1bitの状態を蓄えることができる
- RSフリップフロップ (RS-Flip-Flop)



組み合わせ回路にはなかった、
信号線のループがある

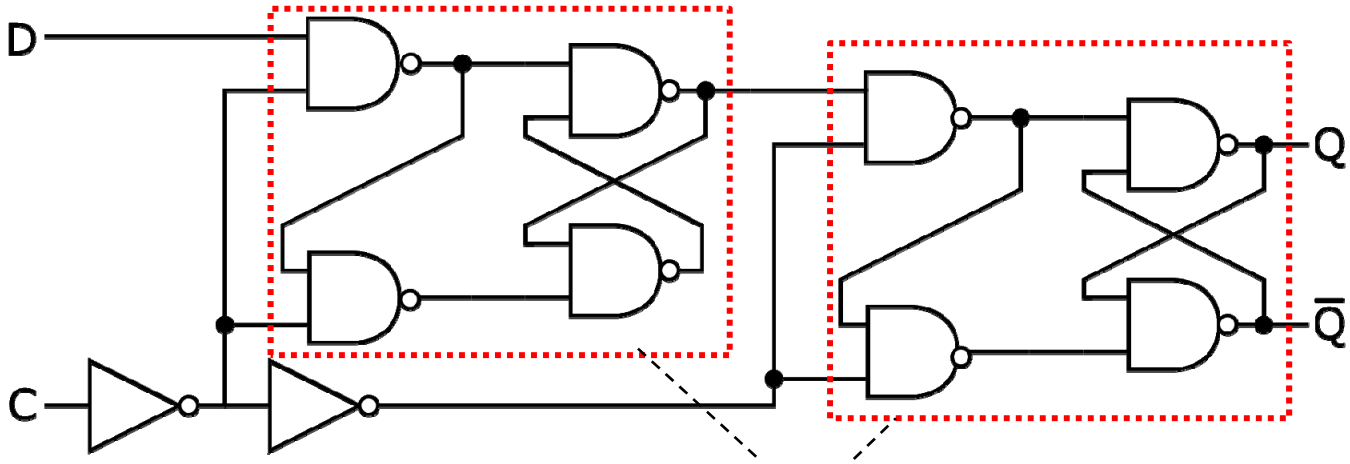
遷移表

		現在	次
R	S	Q	Q
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	?
1	1	1	?

Qが「状態」、$\overline{Q}$はその反転

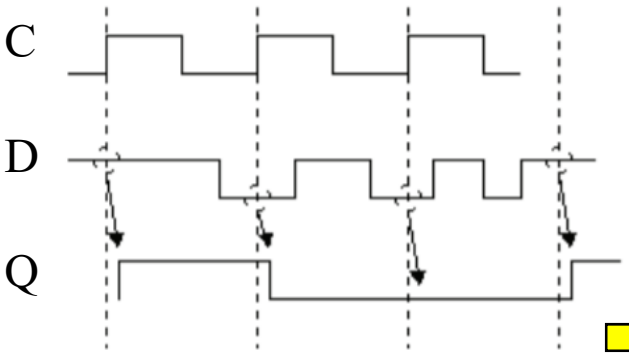
- R=S=0 → Q変化なし
- S=1 → Qが1になる
- R=1 → Qが0になる
- S=R=1 →「禁止」　[Q] なにが起こる？

Dフリップフロップ



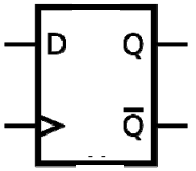
- Qは1bitの状態
- Cが0から1になる瞬間に、D(0 or 1)の値がQに伝播する
  - 上記の瞬間を立ち上がりエッジと呼ぶ
- それ以外の場合は、Qは変化しない

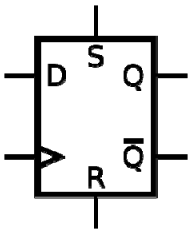
後述の理由により、プロセッサ内ではRSフリップフロップより便利 → レジスタ等に利用



時間の流れ

回路記号

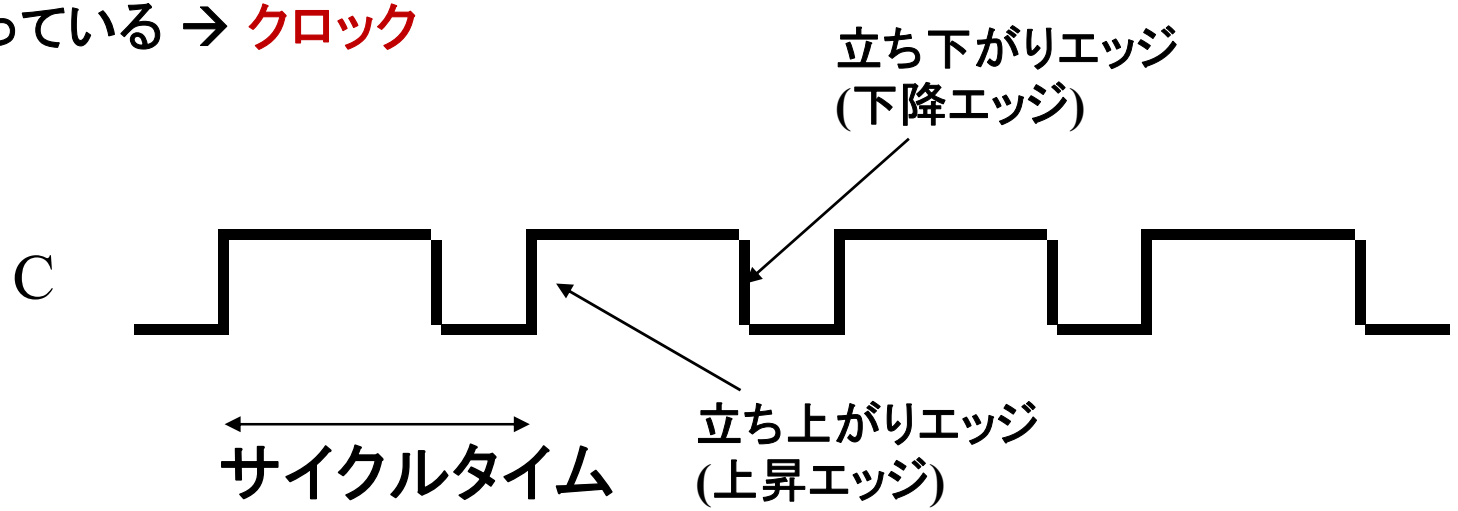




Reset/Set機能
を持つものも

クロックの考え方

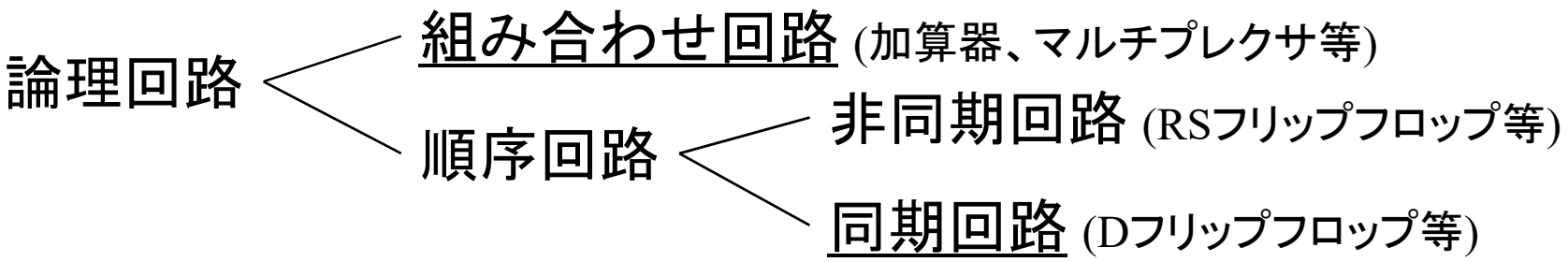
- Dフリップフロップ等に与えられる入力Cは、常に0と1の切り替わりが起こっている → クロック



現在のほぼ全てのプロセッサは、回路全体をつかさどるクロックを用いている

- クロック周波数 = 1sec / サイクルタイム
- 現在の多くのプロセッサでは1～3GHz程度 → なぜもっと上げられないのか

非同期回路と同期回路



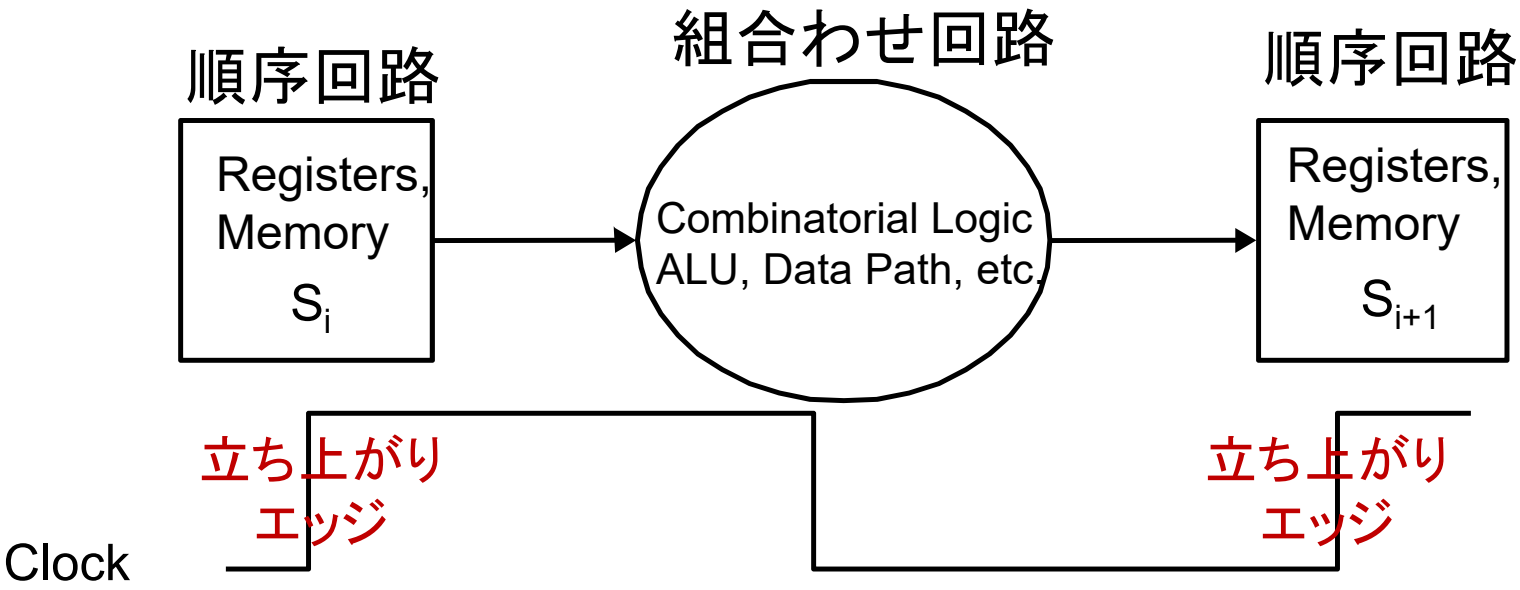
順序回路は、さらに2種類に分類

- 非同期回路(Asynchronous Circuits)：クロックがなく、現在と入力の状態に応じて次の状態に遷移
  - 回路の物理的限界まで高速動作するが、ノイズに弱い、設計困難
- 同期回路(Synchronous Circuits)：クロック(0,1)があり、クロックが状態遷移した際の入力のみが有効
  - クロックが遷移するまでは入力の値は状態遷移には無関係
  - ノイズに強いが、クロック周波数を無限に上げることはできない
  - ほとんどのプロセッサは同期回路として設計

※全ての論理回路は信号値を離散化している。同期回路は、時間も離散化したものと言える

クロックを用いたプロセッサの考え方

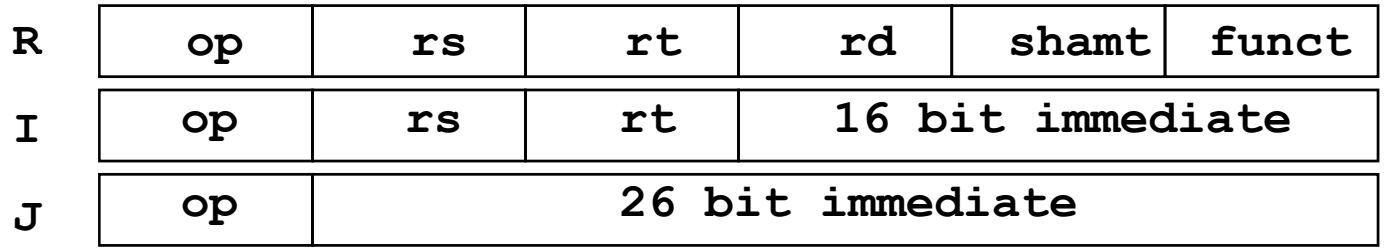
- 典型的な機能ユニット間の関係
  - 状態を表す回路要素の値を読み、
  - 何らかの組合せ回路を通して計算をさせ、
  - 一つ以上の状態を表す(別な)要素回路に結果を書き込む
- 組み合わせ回路内で起こりうる(ゲート遅延などにより)ノイズを吸収
- ただし、サイクルタイムがクリティカルパスの遅延より短いと、動作破綻



いよいよMIPSプロセッサ内部設計へ

復習：MIPSの命令コード

- 1命令=32bit。3通りの命令形式




```
add $s1,$s2,$s3      R     $s1 = $s2 + $s3
sub $s1,$s2,$s3      R     $s1 = $s2 - $s3
slt $s1,$s2,$s3      R     $s1 = if $s2<$s3 then 1 else 0
lw $s1,100($s2)      I     $s1 = Memory[$s2+100]
sw $s1,100($s2)      I     Memory[$s2+100] = $s1
beq $s4,$s5,L        I     jump to L if $s4 = $s5
                           (PC = PC + (16bit imm) x 4)
j L                  J     jump to L
                           (PC = (26bit imm) x 4)
```

(上記の命令 + and + or – j)に対応できるプロセッサを考える

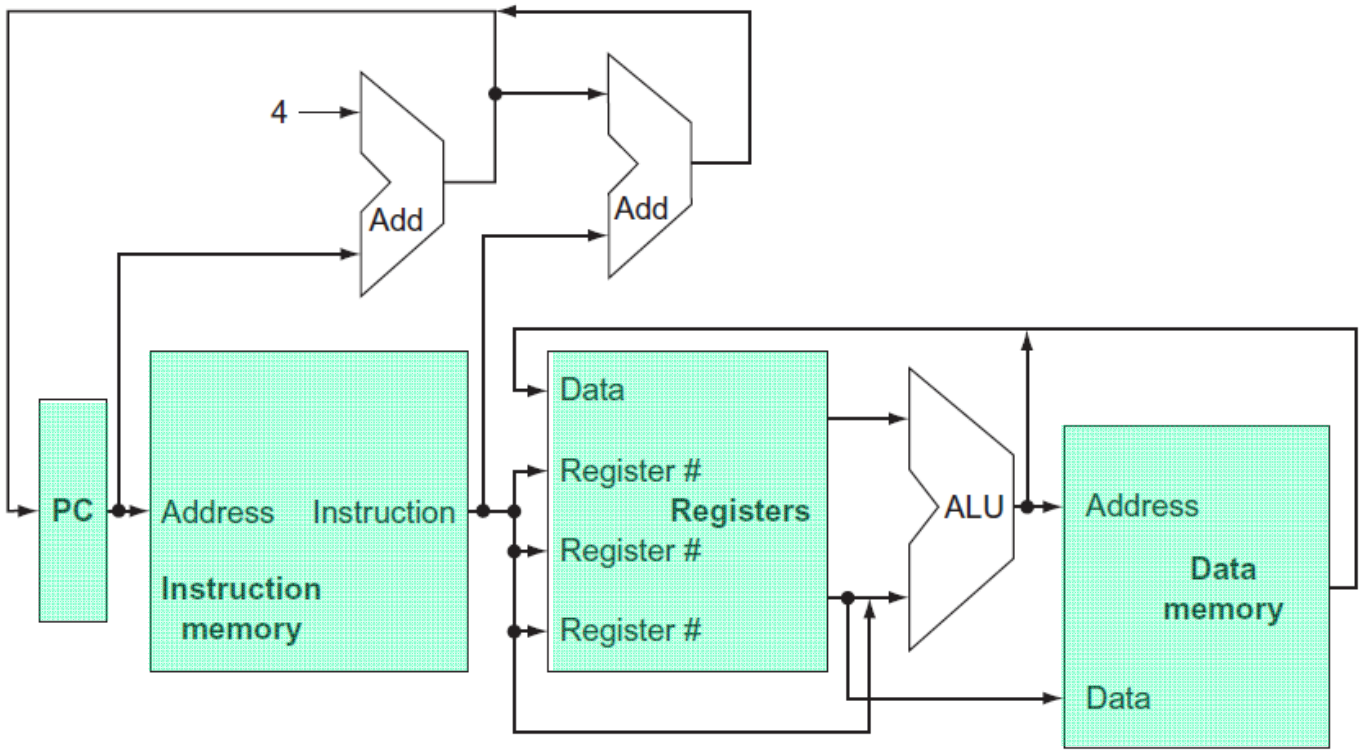
プロセッサの基本動作

- **MIPSプロセッサは以下のような命令サイクルを繰り返す:**
  - **プログラムカウンタ(PC)がさすメモリアドレスから命令を読みだす**
    - 命令フェッチと呼ばれる
  - **指定されたレジスタの値を読み出す (lw, jを除く)**
  - **命令に従って、どのような操作をするかを決定する**
    - (必要に応じて)ALUを使う
    - 指定されたレジスタ・メモリ・PCを書き換える

    } 命令種類によって異なる
  - **PC=PC+4, 一番上へ戻る**

- **レジスタ・メモリ・PCは状態を持つ** ➔ 順序回路が必要

MIPSプロセッサ実装の概観



- 主な機能ユニットと、主な接続のみ示す
- なぜ様々な命令が実行可能か？

  `add $s1,$s2,$s3`　`lw $s1,8($s2)`　`beq $s1,$s2,Label ...`
- この図には何が足りないか？

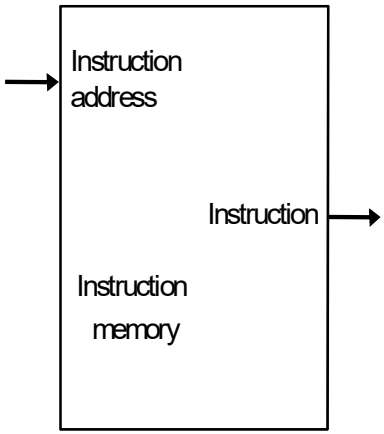
データパス(datapath)と制御(control)

- プロセッサには、複数の機能ユニットが含まれる
  - 機能ユニットは、状態を持つものと、持たないものに分かれる
    - 状態無: ALUなど
    - 状態有: レジスタ、メモリなど

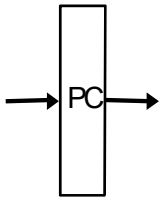
- 機能ユニット間の接続は、以下の2種類
  - データパス(datapath): プロセッサ内のデータの流れ
  - 制御(control): データパスにどの機能ユニットのデータが、どのタイミングで流れるかを制御する

    ⇒前ページの図にはまだ無い

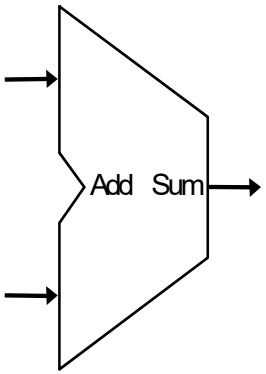
MIPSプロセッサ(簡易版)の中の機能ユニット



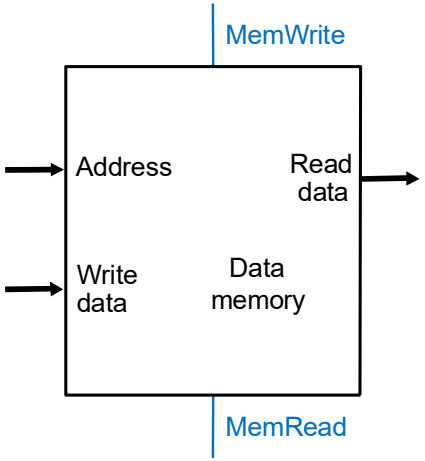
Instruction memory



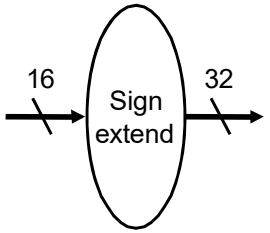
Program counter



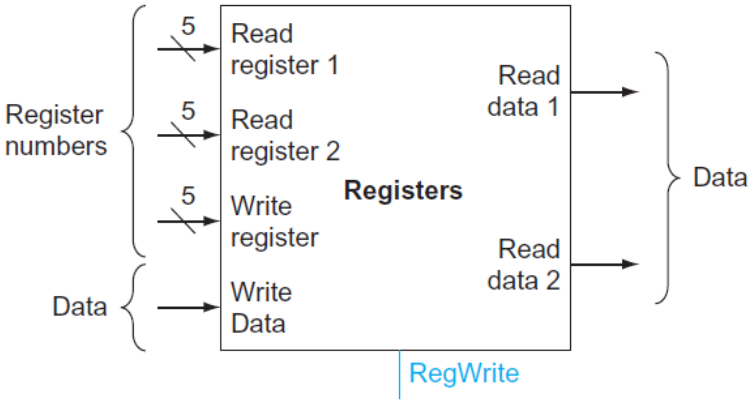
Adder



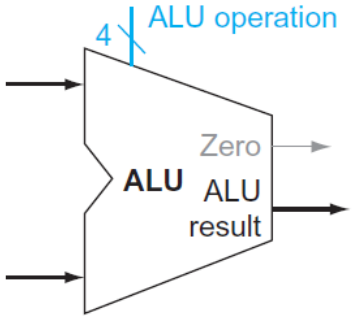
Data memory



Sign extension



Register file



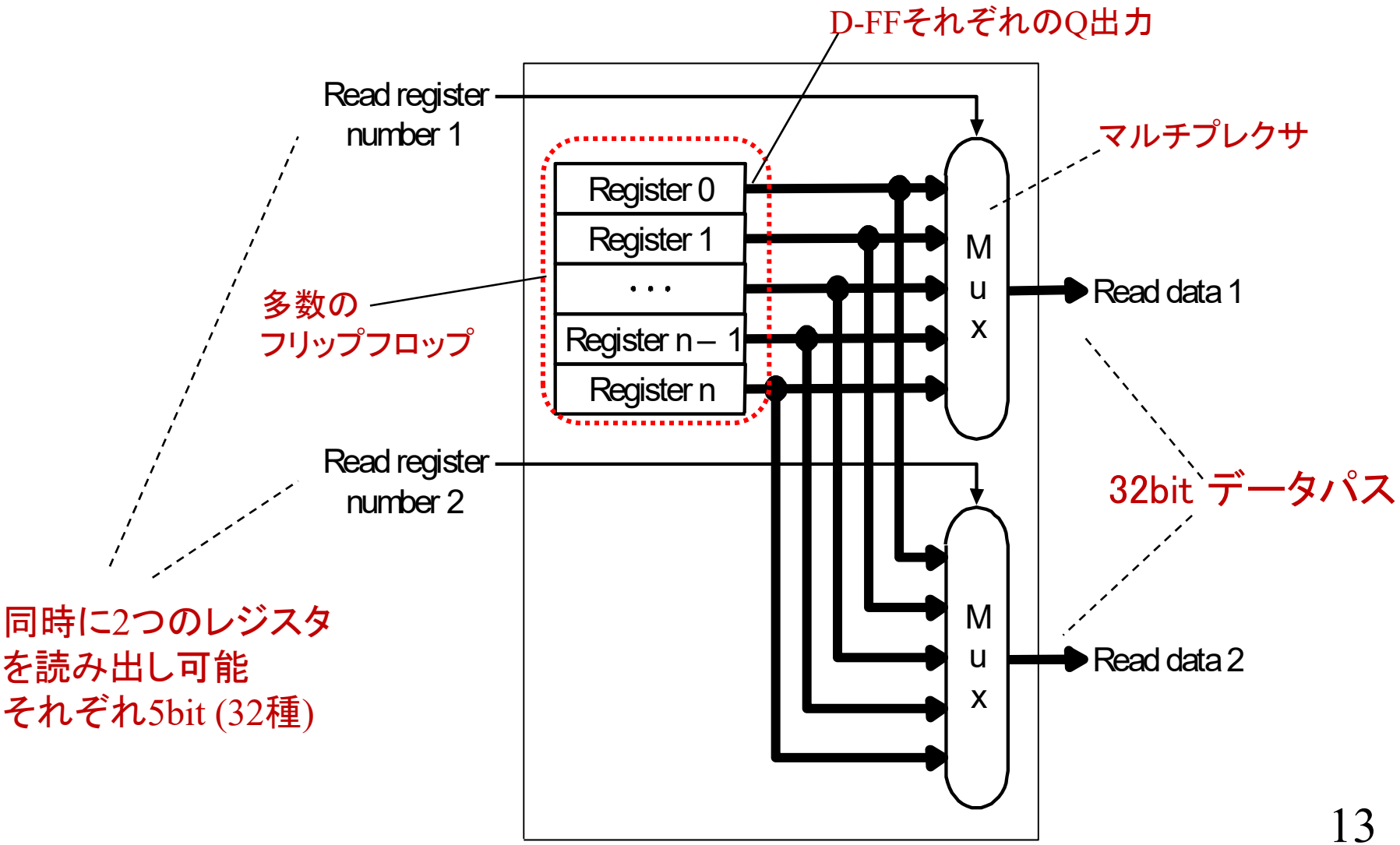
ALU

機能ユニット間の接続:

- **データパス (黒線)**
- **制御 (青線)**

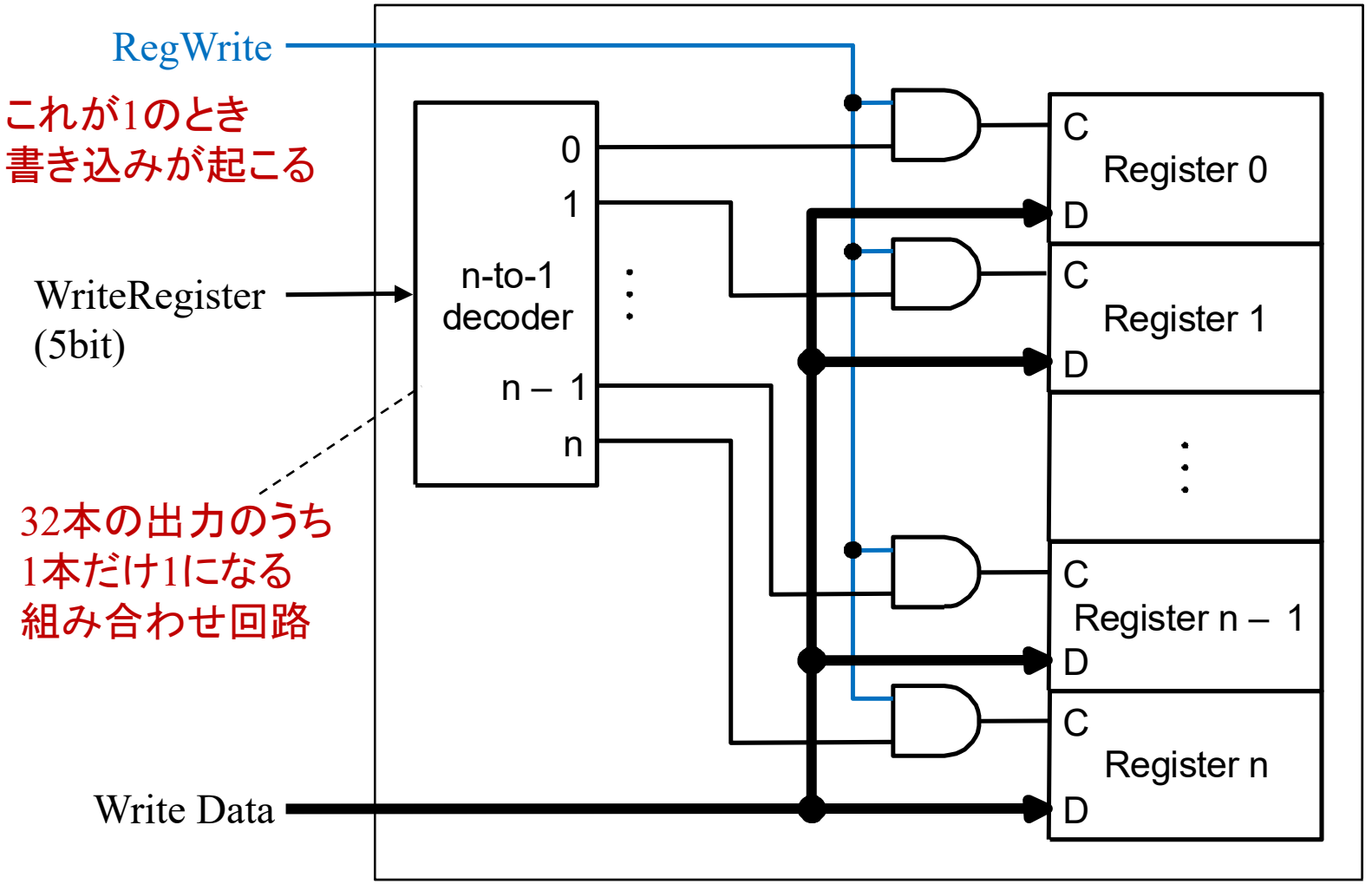
レジスタファイル: レジスタの集合 (1) 読み出し

- 32bitレジスタ×32個を、32x32個のDフリップフロップ(D-FF)で実現



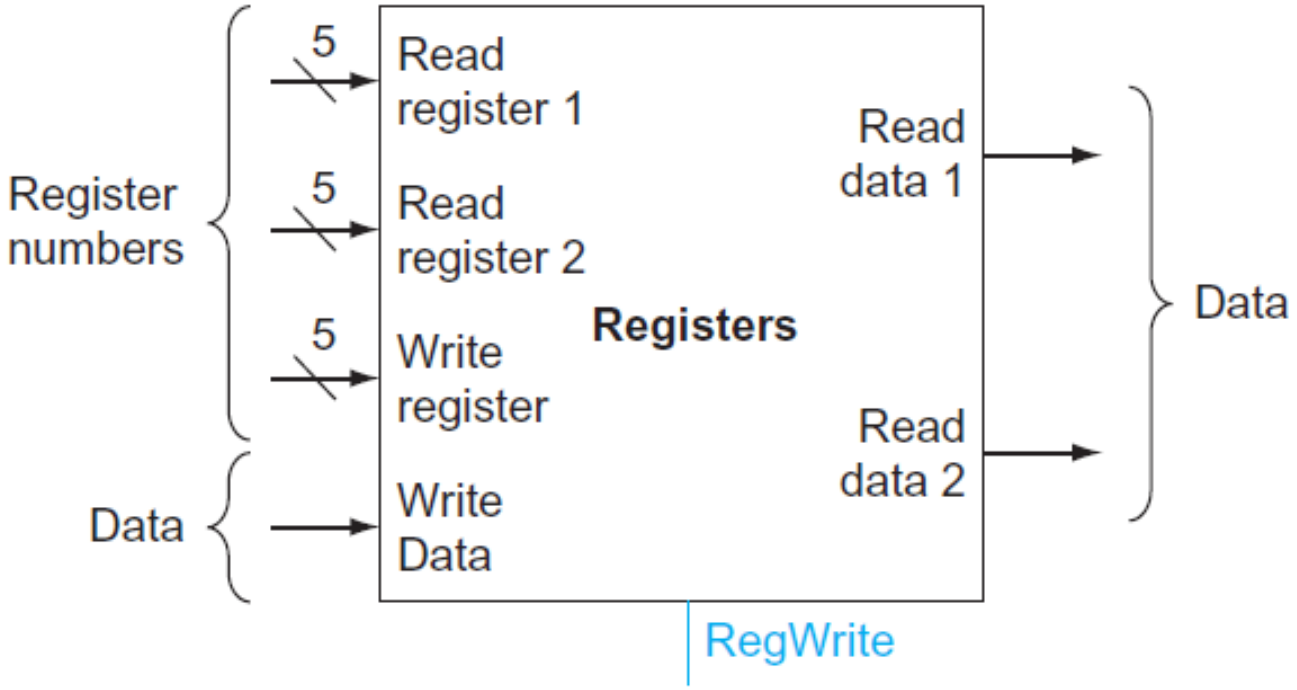
レジスタファイル (2) 書き込み

- 書き込みたいレジスタでだけ、Dフリップフロップの C 入力が1になる



レジスタファイル (3): 全体の入出力

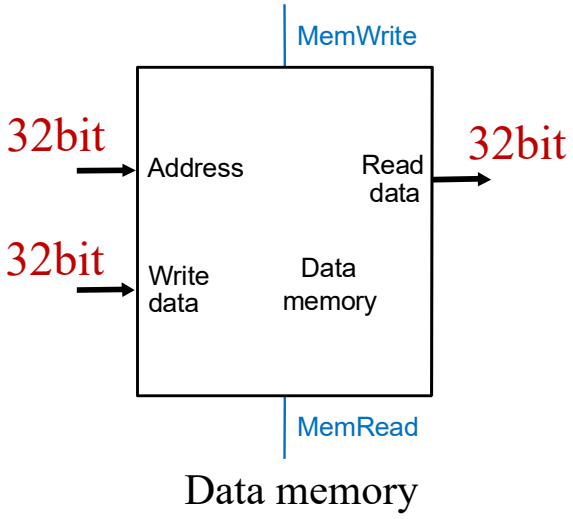
- 読み出し・書き込みの両方に対応するために、以下のような入出力を持つ



メモリ

- 32bitのアドレスと制御信号を入力し、読み出しか書き込みを行えるユニット
- 多くの場合はDRAMと呼ばれる部品
  - 1bitデータ蓄積を、Capacitor (コンデンサ)で行うもの

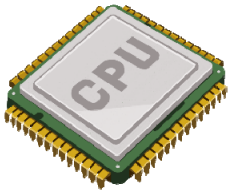
  ⇔ SRAM: フリップフロップを使う。キャッシュメモリなどに利用

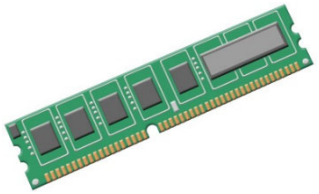


ちなみに実際の通常の計算機では:

- 命令メモリとデータメモリに分かれていない
- メモリはプロセッサの外部に置かれる

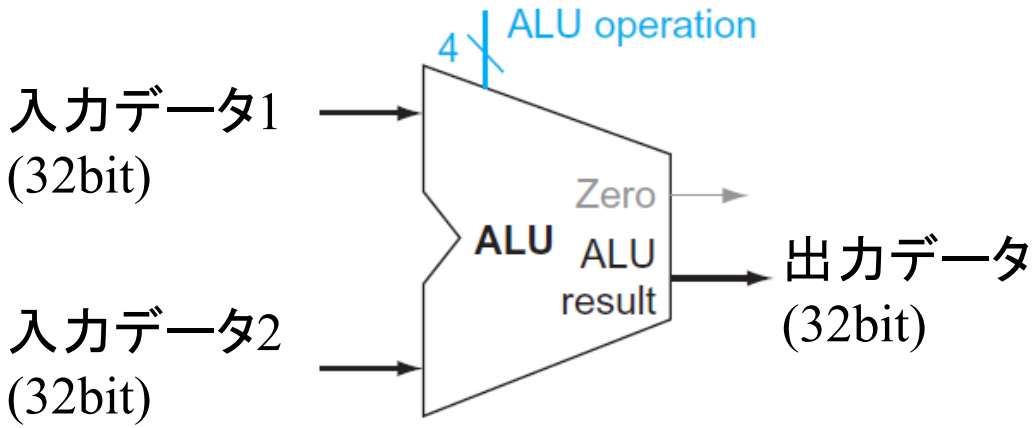
が、今は気にしない





ALU (インタフェースのみ)

- ALUの中身は前回の通り
- ALU operation (4bit)には、演算の種類を指定する

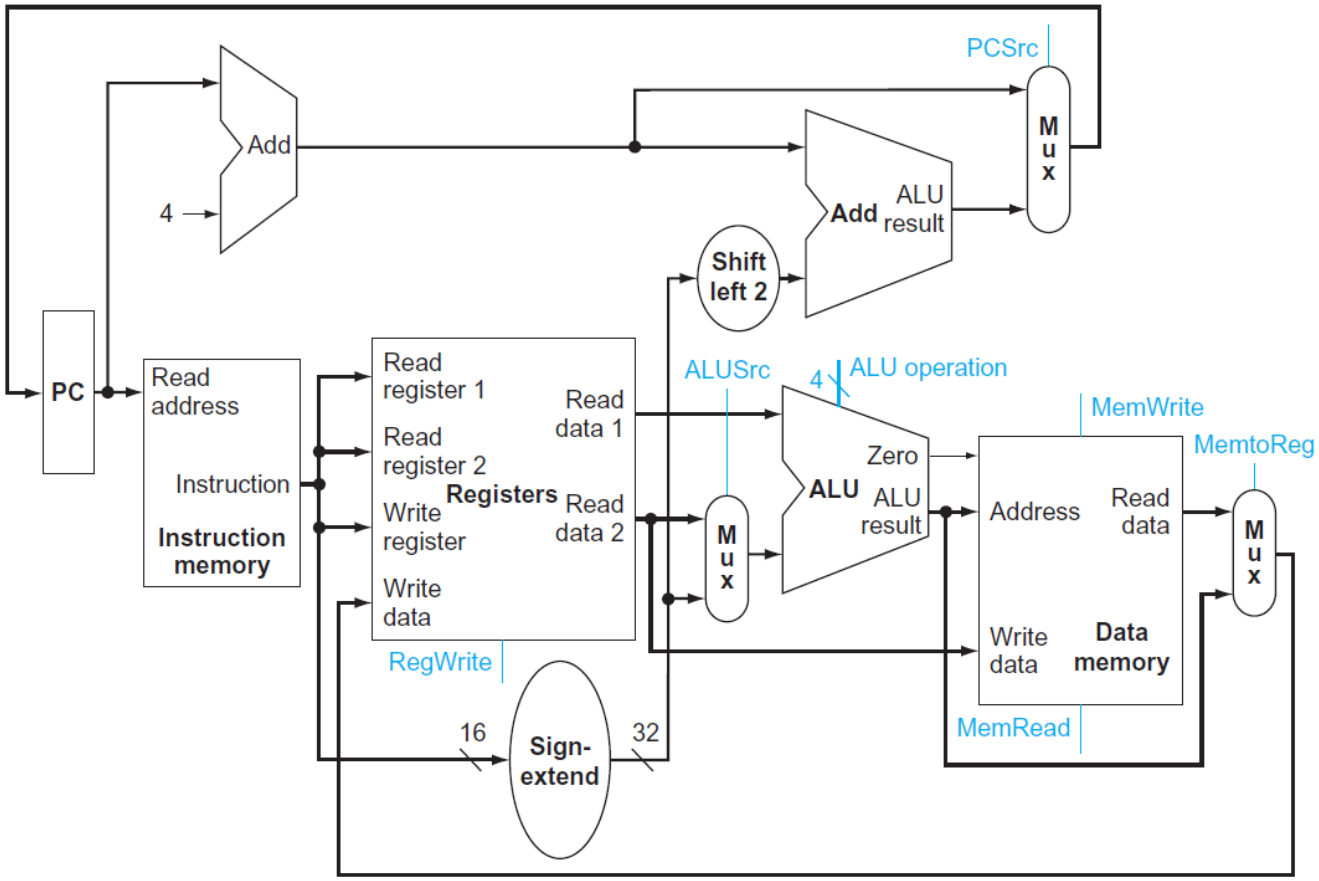


0000 and
0001 or
0010 add
0110 sub
0111 set less than

※この表はISAのように
規約があるわけではないので、
選択肢は一つではない。
この表はPatterson本の例である

機能ユニット間をデータパスで接続

- データパスの「合流」の個所にはマルチプレクサが必要
  - どちらの入力を出力に流したいのか？



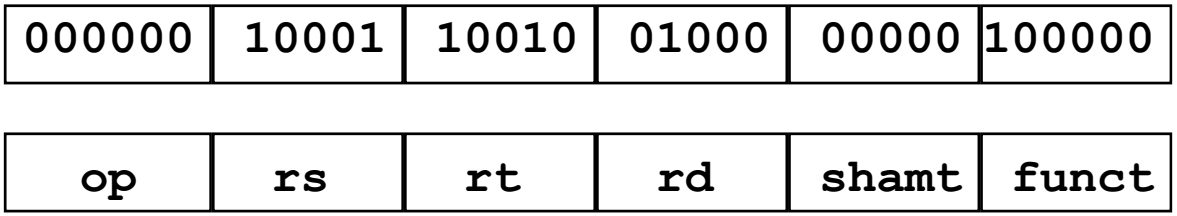
- データパス(黒線)はつながったが、この図ではまだ「制御」(青線)がつながっていない

制御

- データパス上のデータの流れを制御(マルチプレクサの入力)
- それぞれの機能ユニットがどのような操作を行うか？
  - ALU: add, sub, and, or, slt…のうちどれを行いたい？
  - レジスタファイル, メモリ: 読み出し? 書き込み?

- 制御の指定内容は、もともと命令の32bitデータに起因するはず
- 例:

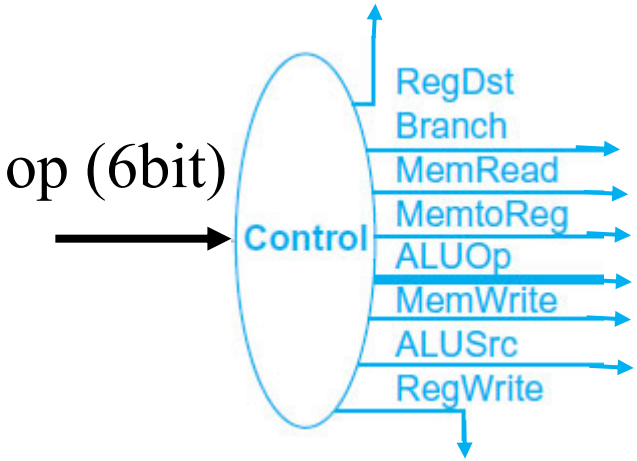
add $8, $17, $18　　R形式の命令フォーマット



注意: この講義では2種類の「制御」が出てきている
(1) 機能ユニット間の制御 → プロセッサ内データの流れを変える(今の話)
(2) beq, j命令などの「制御」命令→プログラムの流れを変える

MIPSプロセッサの制御 (1)

- **RegWrite, MemRead, MemWrite, MemToReg…などの制御線をいつ1に・0にすべきか？ ← 命令コードの”op”の6bitで決まる**
- **以下のような変換を行う回路が新たに必要：制御ユニットと呼ぶ。組み合わせ回路の一つ**



add, and, slt…を含む

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

X: どちらでも関係ない

次ページと関係

lw命令のときはMemReadがswのときはMemWriteが1

MIPSプロセッサの制御 (2): ALUの制御

ALUが動くのは下記のようなとき

(1) R形式の命令(add, and, slt…)のいずれかの実行

- これらの命令はすべてop=000000。funct 6bitで命令種類を区別
  - add: 100000, sub: 100010, slt: 101010…

000000	rs	rt	rd	shamt	funct

(2) beq命令の実行

- beq $s1, $s2, L の際には、($1-$2)の減算が必要

(3) lw, sw命令の実行

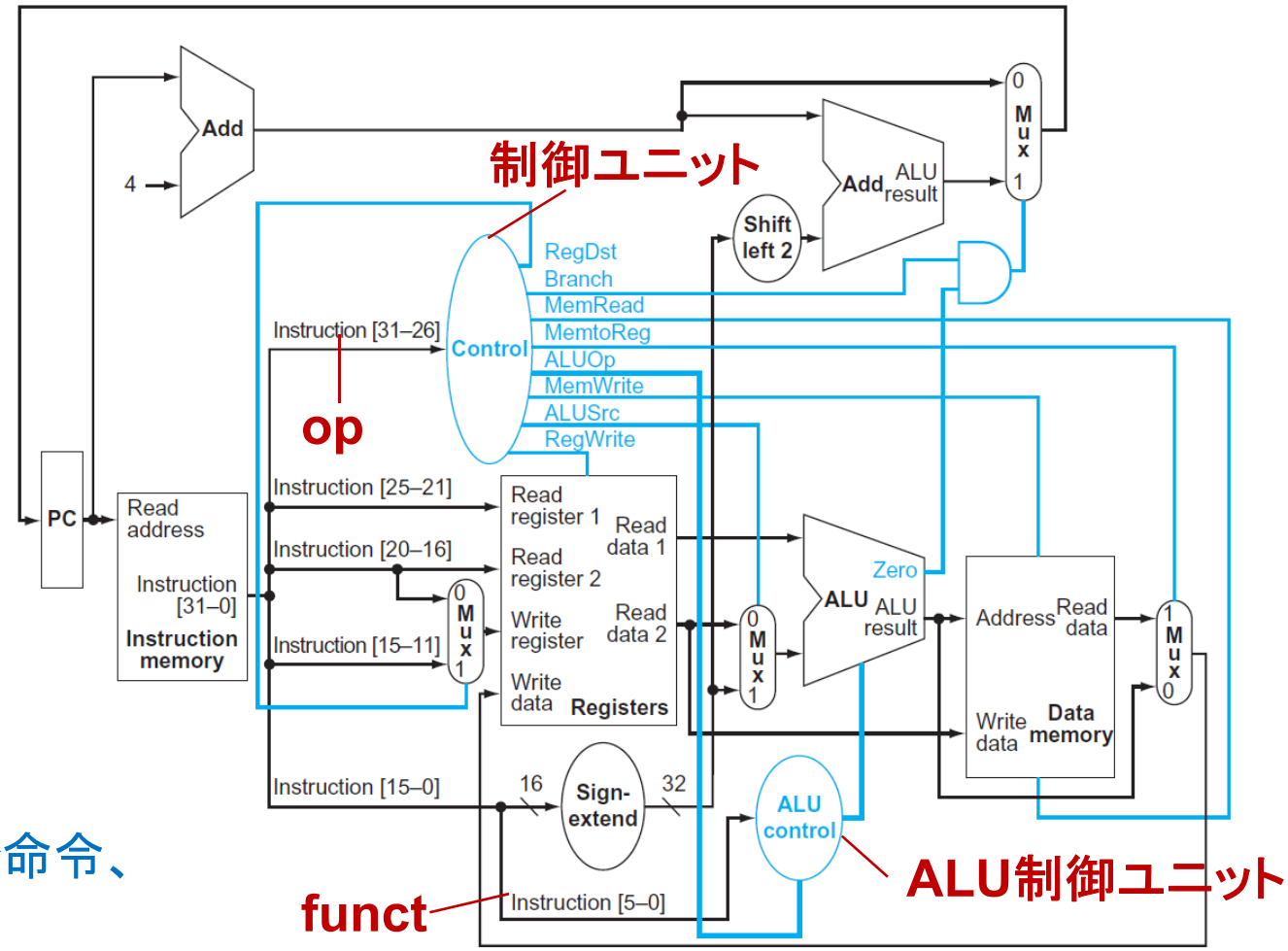
- lw $s1, 100($s2) の際には、($s2+100)の加算が必要

以下のような変換を行う、ALU制御ユニットが必要

入力　　出力 (ALUにとって入力)

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

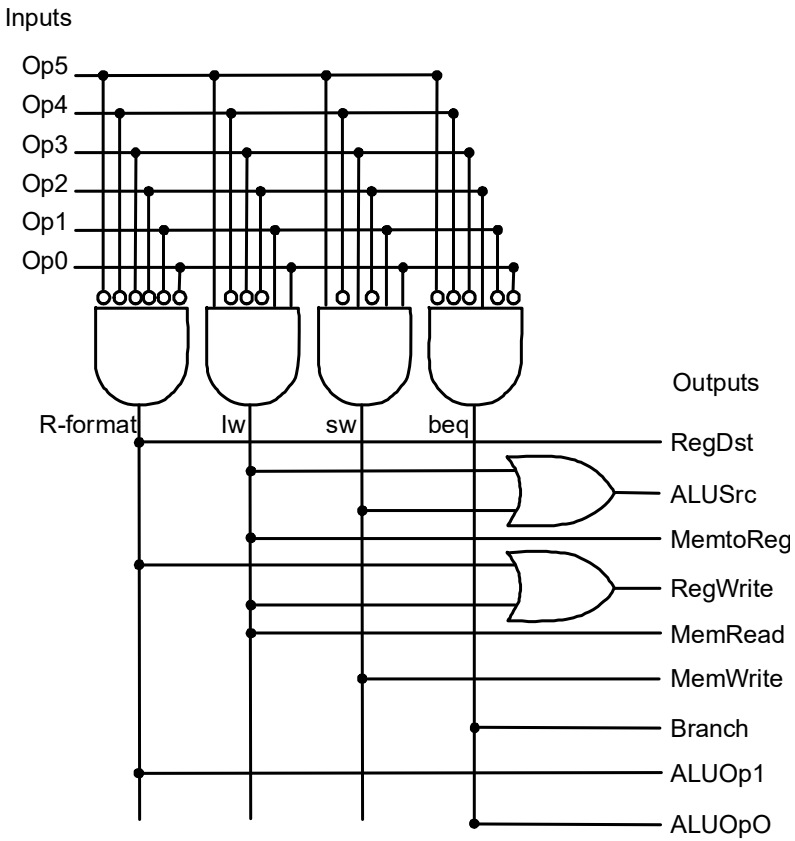
制御を加えた、MIPSプロセッサの(今日の)最終版



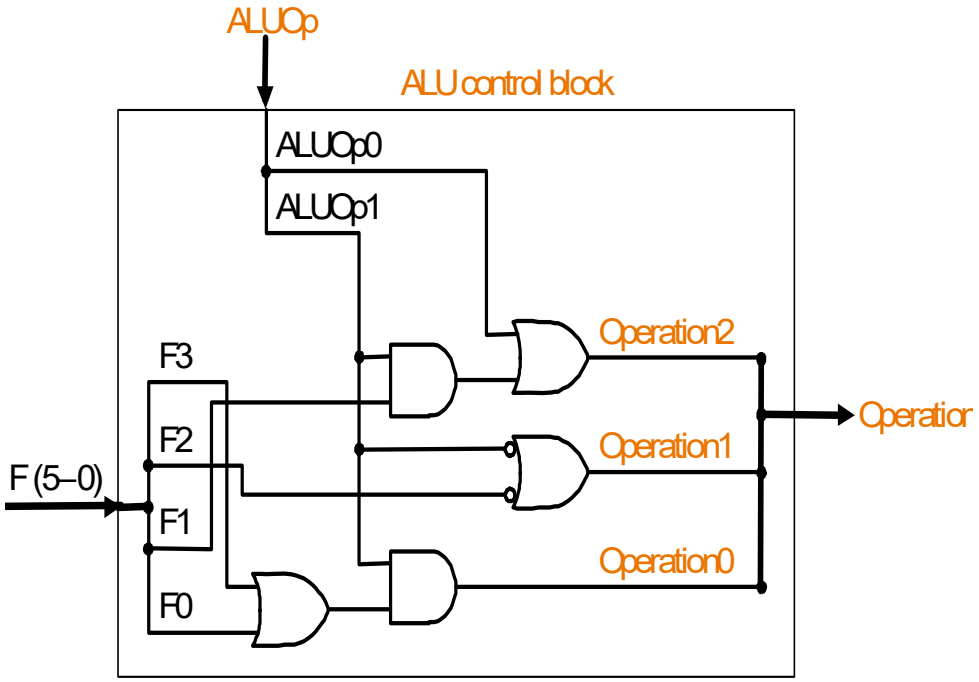
[Q] 演算命令、lw/sw命令、
beq命令それぞれ、
正しく動くか確認しよう

[Q] 現在、j 命令に対応できない。どう改良すればよいか？

制御ユニットの実装例



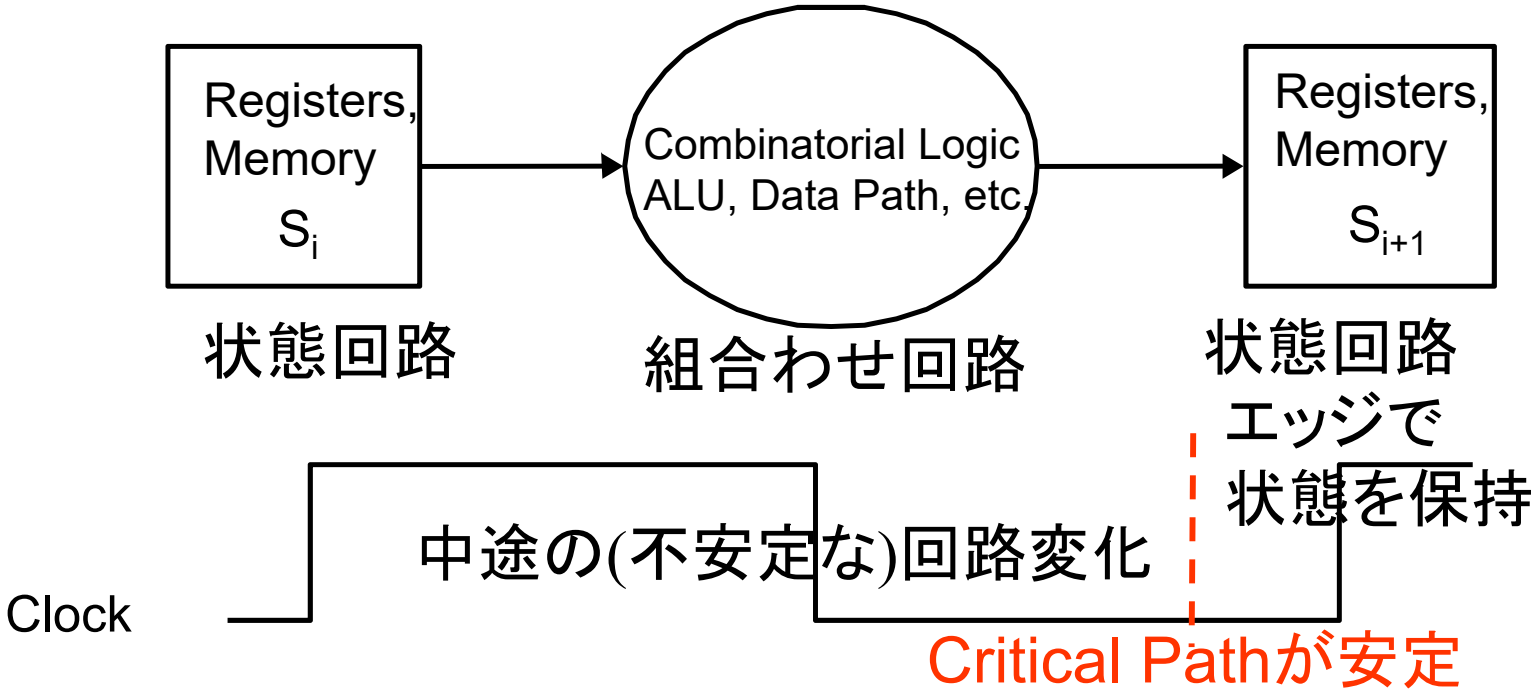
(メイン)制御ユニット



ALU制御ユニット

シングルサイクル設計

- 今回のプロセッサ設計は、「シングルサイクル」と呼ばれる
- すべての組み合わせ回路の出力が安定し、正しい出力を出すまで待つ必要
  - 組み合わせ回路(ALUなど)はすぐには「正しい答え」を出さない
  - そこで、クロック信号と書き込み信号を用いて、いつ書き込むべきか、を決定する
- サイクルタイムは、回路のクリティカルパス(もっとも遅い部分)以上である必要がある → クロック周波数を上げることができない



まとめと改良の方向

- 状態を持たない組み合わせ回路と、状態を持つ順序回路
- MIPSプロセッサ簡略版の実装
  - 機能ブロック (ALU, レジスタ、メモリ…)
  - 機能ブロックどうしの接続
    - データパス
    - 制御

今回の実装はシングルサイクル設計と呼ばれる

- 問題点:
  - クリティカルパスによる性能の低下→浮動小数点演算のように、時間がかかる命令があったらどうなるか?
  - 現在のプロセッサでは使われていない
- より短いサイクルタイム (速いクロック周波数)の実現のためには?
  - 命令によってサイクル数を可変にする → マルチサイクル設計