VLSIアーキテクチャ（2）

坂井　修一
東京大学大学院 情報理工学系研究科
電子情報学専攻

東京大学　工学部
電子情報工学科／電気工学科

- はじめに
- CPUの設計(1)

はじめに

- 本講義の目的
  - VLSIアーキテクチャの基本を学ぶ: 機能 ⇒ VLSI
- 対象者: 工学部4年生以上
- 担当者
  - 坂井修一　プロセッサ → VLSI
  - 池田 誠　アルゴリズム→ VLSI
- 時間・場所
  - 水曜日 8:30 – 10:15
  - 工学部2号館243
- 前提となる知識
  - 電気回路、電子回路
  - ディジタル論理回路
  - 半導体デバイス、VLSI
  - コンピュータアーキテクチャ

教科書、成績

- 教科書
  - 坂井修一『実践　コンピュータアーキテクチャ』(コロナ社)
    坂井部分は教科書通りやります
  - (池田先生の教科書)
- 参考書:　電子デバイス、論理回路、コンピュータアーキテクチャ
  - 坂井修一『論理回路入門』、培風館
  - 坂井修一『コンピュータアーキテクチャ』、コロナ社
  - 電子回路、VLSI
    (池田先生推薦の本)
- http://www.mtl.t.u-tokyo.ac.jp/~sakai/vlsi/
- 成績
  - 実習レポート＋出席

講義の概要と予定

- VLSIアーキテクチャ入門
  - 坂井　4/11
- CPU設計論
  - 坂井　4/18, 5/9, 16, 6/13　（5/16 レポート出題）
- 専用回路設計論
  - 池田　4/25, 5/2, 23, 6/6, 27
- まとめ・将来展望
  - 坂井　6/20
  - 池田　7/4
- 予備 7/11

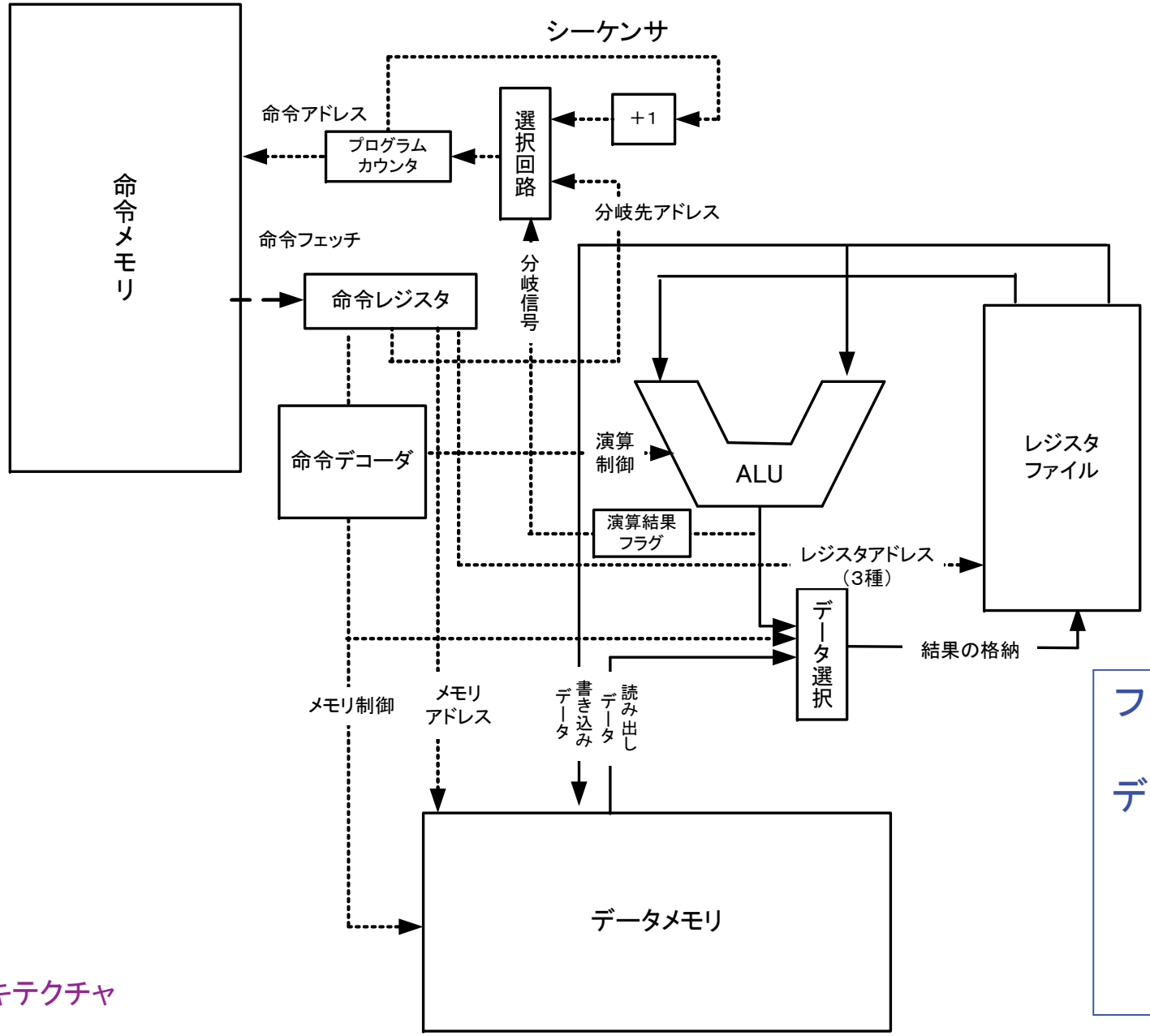
Quartus Primeの実装～シミュレーション

- Intel/Altera Quartus Prime（CADシステム）
  - ダウンロードサイト
    - https://www.altera.co.jp/downloads/download-center.html
    - Quartus Prime Lite (License Free)
      - Quartus本体
      - Modelsim： シミュレータ
  - Tutorialをやってみる
- 本番：『実践 コンピュータアーキテクチャ』参照
  - Verilog HDLで記述
  - パーツの検証： シミュレータはModelsimで
  - 全体の検証
  - 改良、 評価

CPU

- CPU　= Central Processing Unit
  - コンピュータの中央処理装置
  - 32b(64b)CPUのおおざっぱな歴史
    - 1960年代～1970年代: 複数IC、複数ボード
    - 1980年代:　1チップ化の試行錯誤
    - 1990年代:　RISC化、並列化、コモディティ化
    - 2000年代:　マルチコア化、システムLSI化
- 本講義（坂井分）
  - 32ビットCPU　VLSIの設計・改良を行う
  - ３年講義「コンピュータアーキテクチャ」で学習したもの
- ツール
  - アセンブラ:　Perlで作成
  - CAD:　Quartus II (Altera)
  - 回路入力:　Verilog HDL
  - シミュレーション:　Modelsim (Mentor)

基本CPUの構成



フェッチ
↓
デコード
↓
実行
↓
格納

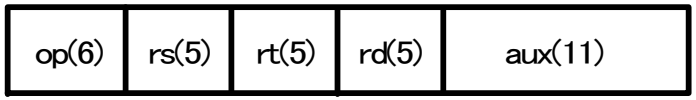
命令

- 命令
  - プロセッサ動作の基本単位
  - 機械語プログラムの基本単位
  - データと同じ2進数で与えられる
    - プログラム格納型　＝　コンピュータの基本
- 命令の種類
  - 算術論理演算命令
    - +, −, *, /
    - AND, OR, NOT, XOR
    - shift
  - メモリ操作命令
    - load, store
  - 分岐命令
    - 無条件分岐命令
    - 条件分岐命令

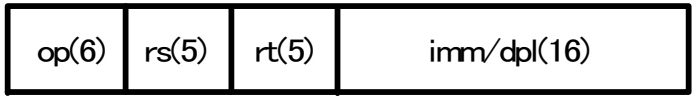
命令形式

- 命令 ＝ 操作 op ＋ 対象
  - 対象 ＝ オペランド
    - ソースオペランド
    - デスティネーションオペランド
  - オペランドとなるもの
    - データレジスタ
    - メモリ語
    - プログラムカウンタ
    - その他のレジスタ
    - 即値

命令語32ビット、命令セットの大きさ64、
レジスタ数32



(1) R型



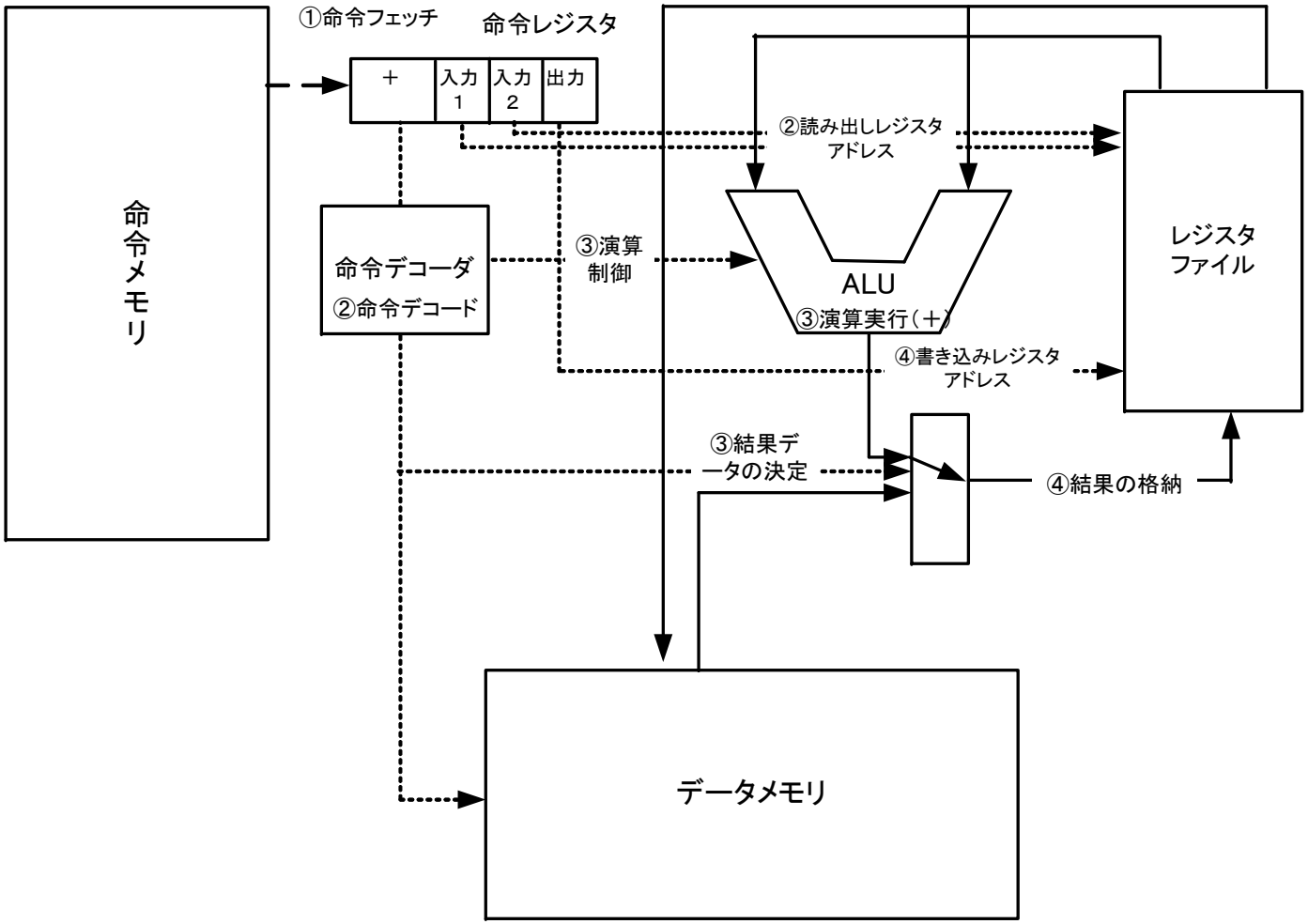
(2) I型

(3) A型

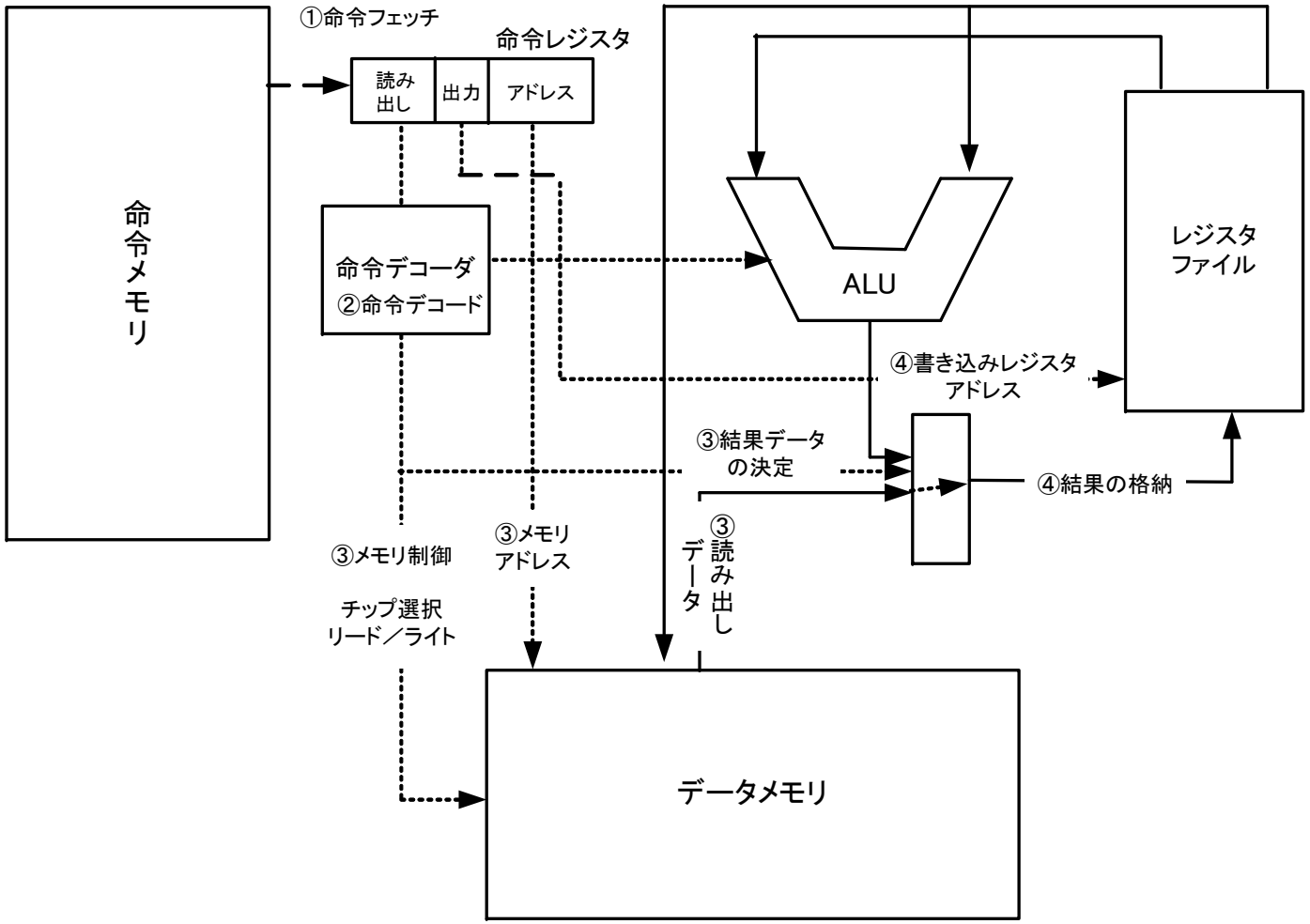
op: 操作コード、 rs, rt, rd: オペランドレジスタ、 aux: 実行細則、
imm/addr: 即値または変位、addr: メモリアドレス

図 3.4 命令のフィールド構成（例）

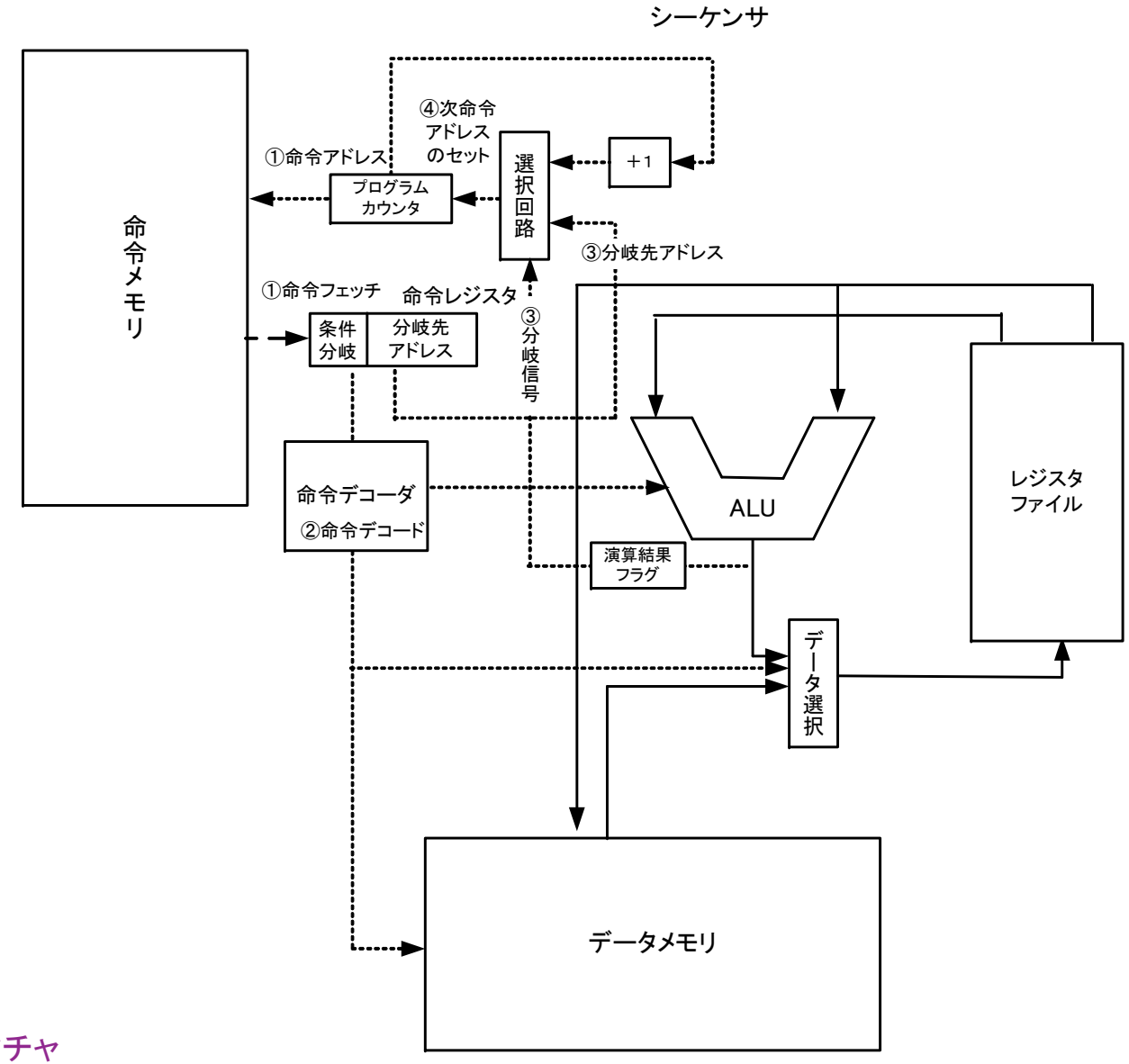
算術論理演算命令の実行サイクル



メモリ操作命令の実行サイクル

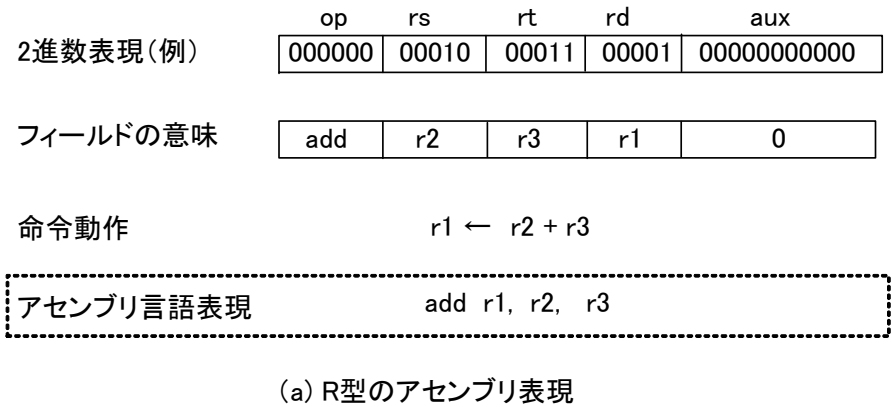


条件分岐命令の実行サイクル

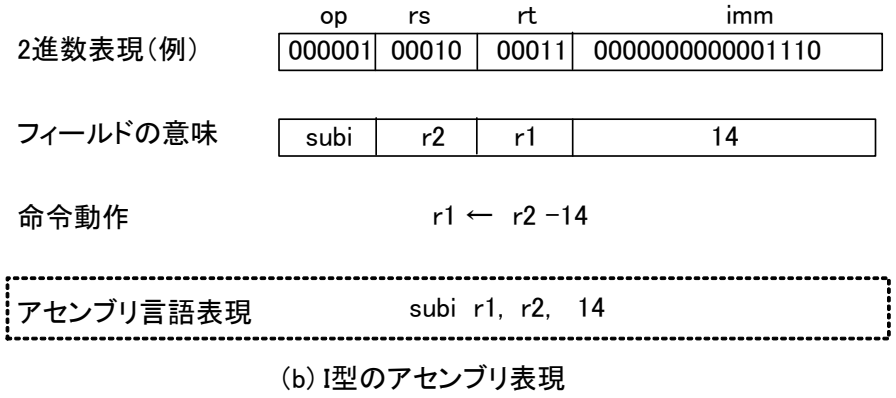


アセンブリ言語

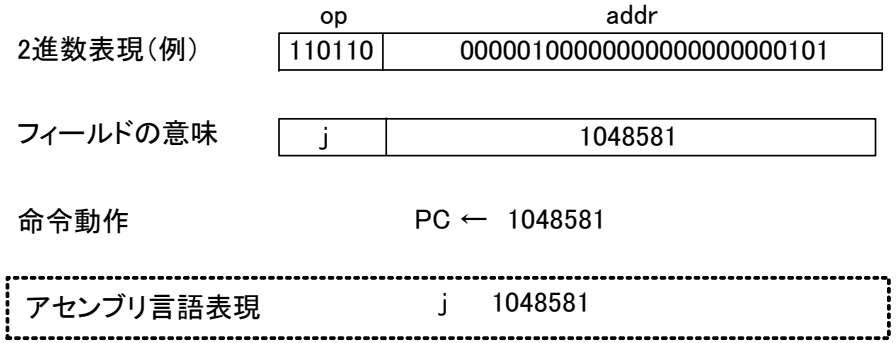
- プログラムの表記
  - 機械語： 読みにくい
- アセンブリ言語
  - 英語に近い記号で表記
  - 機械語と1対1対応
  - ニューモニック
    - OPを英語に近い標記にしたもの



(a) R型のアセンブリ表現



(b) I型のアセンブリ表現



(c) A型のアセンブリ表現

命令セットとニューモニック

ニューモニック	命令形式	OP	AUX等	動作	補注
add	R	0	0	rd <- rs + rt	
addi	I	1	imm	rt <- rs + imm	
sub	R	0	2	rd <- rs - rt	
lui	I	3	imm	rt <- imm << 16	上位16ビットをセット
and	R	0	8	rd <- rs AND rt	ビットごとのAND
andi	I	4	imm	rt <- rs AND imm	即値とのAND
or	R	0	9	rd <- rs OR rt	
ori	I	5	imm	rt <- rs OR imm	
xor	R	0	10	rd <- rs XOR rt	
xori	I	6	imm	rt <- rs XOR imm	
nor	R	0	11	rd <- rs NOR rt	
sll	R	0	16	rd <- rs llshift aux[10,6]	論理左シフト
srl	R	0	17	rd <- rs lrshift aux[10,6]	論理右シフト
sra	R	0	18	rd <-rs arshift aux[10,6]	算術右シフト
lw	I	16	dpl	rt <- mem(rs+dpl)	一語読み出し
lh	I	18	dpl	rt <- mem(rs+dpl)	半語読み出し、符号拡張
lb	I	20	dpl	rt <- mem(rs+dpl)	バイト読み出し、符号拡張
sw	I	24	dpl	mem(rs+dpl) <- rt	一語書き込み
sh	I	26	dpl	mem(rs+dpl) <- rt	半語書き込み、符号拡張
sb	I	28	dpl	mem(rs+dpl) <- rt	バイト書き込み、符号拡張
beq	I	32	dpl	if rs == rt then PC = PC+dpl	等しければジャンプ
bne	I	33	dpl	if rs != rt then PC = PC+dpl	等しくなければジャンプ
blt	I	34	dpl	if rs < rt then PC = PC+dpl	未満であればジャンプ
ble	I	35	dpl	if rs <= rt then PC = PC+dpl	以下であればジャンプ
j	A	40	addr	PC <- addr	絶対番地のジャンプ
jal	A	41	addr	R31 <- PC+4; PC <- addr	ジャンプアンドリンク
jr	R	42		PC <- rs	レジスタ間接ジャンプ

アセンブラ

- アセンブラ
  - アセンブリ言語のプログラムを機械語（2進数）に変換するプログラム
- Perl言語による作成例
  - 教科書　図6.16

```
open(FH, "@ARGV[0]");
 $i = 0;
 # ラベル、カンマ、空白、タブの処理
 while ($line = <FH>) {
 $line =~ s/,/ /g;
 $line =~ s/¥t/ /g;
 $line =~ s/¥:/ : /g;
 $line =~ s/^ +//g;
 chomp($line);
 @instruction = split(/ +/, $line);
 if (@instruction[1] eq ":") {
 $labels{@instruction[0]} = $i;
 }
 $i++;
 }
 close(FH);

 open(FH, "@ARGV[0]");
 $i = 0;
 while ($line = <FH>) {
 # print("$i : ");
 $line =~ s/,/ /g;
 $line =~ s/¥t+/ /g;
 $line =~ s/¥:/ : /g;
 $line =~ s/^ +//g;
 # print($line);
 chomp($line);
 @instruction = split(/ +/, $line);
 if (@instruction[1] eq ":") {#    ラベルのある行をフィールドに分割する
 $op = @instruction[2];
 $f2 = @instruction[3];
 $f3 = @instruction[4];
 $f4 = @instruction[5];
 $f5 = @instruction[6];
 }
```

```
else {# ラベルのない行をフィールド分割する
 $op = @instruction[0];
 $f2 = @instruction[1];
 $f3 = @instruction[2];
 $f4 = @instruction[3];
 $f5 = @instruction[4];
 }
# 機械語の出力
    if ($op eq "add"){p_b(6,0); p_r3($f2, $f3, $f4); p_b(11,0);print("¥n");}
    elsif ($op eq "addi") {p_b(6,1); p_r2i($f2, $f3); p_b(16, $f4);print("¥n");}
    elsif ($op eq "sub"){p_b(6,0); p_r3($f2, $f3, $f4); p_b(11, 2); print("¥n");}
    elsif ($op eq "lui") {p_b(6,3); p_r2i($f2, "r0"); p_b(16, $f3);print("¥n");}
    elsif ($op eq "and"){p_b(6,0); p_r3($f2, $f3, $f4); p_b(11, 8); print("¥n");}
    elsif ($op eq "andi") {p_b(6,4); p_r2i($f2, $f3); p_b(16, $f4);print("¥n");}
    elsif ($op eq "or"){p_b(6,0); p_r3($f2, $f3, $f4); p_b(11,9);print("¥n");}
    elsif ($op eq "ori"){p_b(6,5); p_r2i($f2, $f3); p_b(16, $f4);print("¥n");}
    elsif ($op eq "xor"){p_b(6,0); p_r3($f2, $f3, $f4); p_b(11,10);print("¥n");}
    elsif ($op eq "xori"){p_b(6,6); p_r2i($f2, $f3); p_b(16, $f4);print("¥n");}
    elsif ($op eq "nor"){p_b(6,0); p_r3($f2, $f3, $f4); p_b(11,11);print("¥n");}
    elsif ($op eq "sll"){p_b(6,0); p_r3($f2, $f3, "r0"); p_b(5, $f4); p_b(6,16);print("¥n");}
    elsif ($op eq "srl"){p_b(6,0); p_r3($f2, $f3, "r0"); p_b(5, $f4); p_b(6,17);print("¥n");}
    elsif ($op eq "sra"){p_b(6,0); p_r3($f2, $f3, "r0"); p_b(5, $f4); p_b(6,18);print("¥n");}
    elsif ($op eq "lw"){p_b(6,16); p_r2i($f2, base($f3)); p_b(16, dpl($f3));print("¥n");}
    elsif ($op eq "lh"){p_b(6,18); p_r2i($f2, base($f3)); p_b(16, dpl($f3));print("¥n");}
    elsif ($op eq "lb"){p_b(6,20); p_r2i($f2, base($f3)); p_b(16, dpl($f3));print("¥n");}
    elsif ($op eq "sw"){p_b(6,24); p_r2i($f2, base($f3)); p_b(16, dpl($f3));print("¥n");}
    elsif ($op eq "sh"){p_b(6,26); p_r2i($f2, base($f3)); p_b(16, dpl($f3));print("¥n");}
    elsif ($op eq "sb"){p_b(6,28); p_r2i($f2, base($f3)); p_b(16, dpl($f3));print("¥n");}
    elsif ($op eq "beq") {p_b(6,32); p_r2b($f2, $f3); p_b(16, $labels{$f4}-$i-1);print("¥n");}
    elsif ($op eq "bne") {p_b(6,33); p_r2b($f2, $f3); p_b(16, $labels{$f4}-$i-1);print("¥n");}
    elsif ($op eq "blt") {p_b(6,34); p_r2b($f2, $f3); p_b(16, $labels{$f4}-$i-1);print("¥n");}
    elsif ($op eq "ble") {p_b(6,35); p_r2b($f2, $f3); p_b(16, $labels{$f4}-$i-1);print("¥n");}
    elsif ($op eq "j") {p_b(6,40); p_b(26, $labels{$f2});print("¥n");}
    elsif ($op eq "jal"){p_b(6, 41); p_b(26, $labels{$f2});print("¥n");}
    elsif ($op eq "jr"){p_b(6, 42); p_r3("r0", $f2, "r0"); p_b(11, 0);print("¥n");}
    else {print("ERROR: Illegal Instruction¥n");}
    $i++;
 }
 close(FH);
```

```
sub p_b{#  $numを2進数$digitsに変換して出力
($digits, $num) = @_;
if ($num >= 0) {
printf("%0".$digits."b_", $num);
} else {
print(substr(sprintf("%b ", $num), 32-$digits));
}
}

sub p_r3{#    R型のレジスタ番地を出力
($rd, $rs, $rt) = @_;
$rs =~ s/r//; p_b(5, $rs);
$rt =~ s/r//; p_b(5, $rt);
$rd =~ s/r//; p_b(5, $rd);
}

sub p_r2i{#    I型のレジスタ番地を出力
($rt, $rs) = @_;
$rs =~ s/r//; p_b(5, $rs);
$rt =~ s/r//; p_b(5, $rt);
}

sub p_r2b{# 条件分岐で比較するレジスタ番地を出力
($rs, $rt) = @_;
$rs =~ s/r//; p_b(5, $rs);
$rt =~ s/r//; p_b(5, $rt);
}

sub base{#     ベースアドレスレジスタの番地を返す
($addr) = @_;
$addr =~ s/.*¥(//;
$addr =~ s/¥)//;
return($addr);
}

sub dpl{# 変位を返す
($addr) = @_;
$addr =~ s/¥(.*¥)//;
return($addr);
}
```

宿題2

- Perlシステムを適切なWWWサイトからダウンロードしてインストールせよ。Windows系のコンピュータであれば、http://www.activestate.com/Products/activeperl/index.mhtmlなどがダウンロードサイトとなる。

- 前頁のアセンブラを実装せよ。そのさい、Perlの基本的な文法を理解せよ。

- 上で作ったアセンブラで、自分で作った短いプログラムを機械語に翻訳せよ。１＋１＝２などでよい。

- 来週までにやっておくこと。提出は無し。ただし、レポート課題のためには必須の作業となる。