3 命令セット (1)

♣ MIPS を例に, コンピュータの命令セットと 機械語 がどのようなものか知る

♣ 例を通じて, 命令がどのように 利用 されるかを知る

3.1 MIPS の演算命令

3.1.1 MIPS のレジスタ

- MIPS[1]は主な演算やデータ操作において 32 bit (= 4 バイト) のデータを扱う[2]
  - この単位を語 (word) と呼ぶ
  - 符号付き整数は 2 の補数 表現
- MIPS のレジスタ (すべて 32 bit)

PC	プログラムカウンタ (program counter)
$0〜$31	32 個の 汎用 レジスタ (注 1)　ただし, $0 だけは特殊なレジスタ (注 2)

(注 1) 汎用レジスタ (general purpose register)

コンピュータによっては, プログラムカウンタの他にも特殊な用途専用のレジスタを持つことがある (アキュムレータ, インデックスレジスタ, スタックポインタ, フレームポインタ等). これに対し, 「どの目的にも使用できるレジスタ」を多数持つ構成が現在の主流であり, このようなレジスタを汎用レジスタと呼ぶ

(注 2) $0 はゼロレジスタ (zero register) と呼ばれ, 次のような性質を持つ

- $0 を読み出すと, 値は常に 0 である
- $0 に値を書き込んでも, 何も起こらない

演算の記法

- `*` ⋯ 乗算　`/` ⋯ 整数除算　`%` ⋯ 整数剰余
- `&` ⋯ ビット毎の論理積 (and) (`b0101 & b0011 = b0001`)
- `|` ⋯ ビット毎の論理和 (or) (`b0101 | b0011 = b0111`)
- `~` ⋯ ビット毎の論理否定 (not) (`~ b0101 = b1010`)
- $\oplus$ ⋯ ビット毎の排他的論理和 (exclusive-or) (`b0101` $\oplus$ `b0011 = b0110`)
- $<<_L$ ⋯ 論理左シフト (`b001101` $<<_L$ `1 =` b011010)
- $>>_L$ ⋯ 論理右シフト (`b101100` $>>_L$ `1 =` b010110)
- $>>_A$ ⋯ 算術右シフト (`b101100` $>>_A$ `1 =` b110110)
- $zx(x)$ ⋯ x の 32bit への ゼロ拡張 (zx(`xC19A`) `= x0000C19A`　(常に 0⋯0 を付加))
- $sx(x)$ ⋯ x の 32bit への 符号拡張 (sx(`xC19A`) `= xFFFFC19A`, sx(`x419A`) `= x0000419A`)

[1] MIPS Computer Systems および MIPS Technologies が開発した, マイクロプロセッサのアーキテクチャ. “Microprocessor without Interlocked Pipeline Stages” の略.

[2] その後の拡張で 64bit データを扱うものも現れているが, 本講義では 32bit 版のみを扱う.

|| … 連結 (b1010 || b1111 = b10101111)

$c\,?\,a:b$ … c が真なら a, 偽なら b ((5==4+1)?3:9 = 3, (5<6-2)?3:9 = 9)

$R_{x:y}$ … レジスタ R の $x \sim y$ bit 目 (レジスタは, 一番左が 31bit 目で, 一番右が 0bit 目)

例えば, $PC_{31:28}$ はプログラムカウンタの 上位 4 bit を表す

Mem[a,b] … 主記憶の a 番地 から始まる b バイト のデータ

3.1.2 レジスタ同士の演算命令

- 加算命令の例

  `add $21,$7,$9` (add 命令)

  - (動作) \$21 = \$7 + \$9 , PC = PC + 4
    - ∗ 7 番レジスタの値と 9 番レジスタの値を加算し, 結果を 21 番レジスタに入れる
    - ∗ PC が 次の命令 を指すようにする[3]
  - \$7=12, \$9=22, PC=1024 のときに add \$21,\$7,\$9 を実行 → \$21= 34 , PC= 1028
  - \$21 や \$7 や \$9 等を オペランド (operand ; 被演算値) と呼ぶ
- 加算命令の一般形

  `add Rd, Rs, Rt`

  - Rd, Rs, Rt はレジスタ (\$0∼\$31 のいずれか)
  - (動作) $Rd = Rs + Rt$, PC = PC + 4
- ゼロレジスタ \$0 に注意 (普通のレジスタではない!)
  - \$0 の値は常に 0

    add \$21,\$7,\$0 を実行 → \$21 = \$7 となる (レジスタの コピー)
  - \$0 に値を書き込むことはできない

    add \$0,\$5,\$13 を実行 → 何も起こらない (PC の値は 4 増えるが)

  ☆ 何も演算を行わない命令を NOP (no operation) と言う
- レジスタ同士の演算命令

add Rd, Rs, Rt	$Rd = Rs + Rt$, PC = PC + 4	加算（符号付き）
sub Rd, Rs, Rt	$Rd = Rs - Rt$, PC = PC + 4	減算（符号付き）
addu Rd, Rs, Rt	$Rd = Rs + Rt$, PC = PC + 4	加算（符号無し）(注 1)
subu Rd, Rs, Rt	$Rd = Rs - Rt$, PC = PC + 4	減算（符号無し）(注 1)
and Rd, Rs, Rt	$Rd = Rs \,\&\, Rt$, PC = PC + 4	ビット毎の論理積
or Rd, Rs, Rt	$Rd = Rs \,|\, Rt$, PC = PC + 4	ビット毎の論理和
xor Rd, Rs, Rt	$Rd = Rs \oplus Rt$, PC = PC + 4	ビット毎の排他的論理和
nor Rd, Rs, Rt	$Rd = \tilde{}(Rs \,|\, Rt)$, PC = PC + 4	ビット毎の論理和否定
slt Rd, Rs, Rt	$Rd = (Rs < Rt)$? 1 : 0, PC = PC + 4	比較「小なり」（符号付き）(注 2)
sltu Rd, Rs, Rt	$Rd = (Rs < Rt)$? 1 : 0, PC = PC + 4	比較「小なり」（符号無し）(注 2)

[3] 1 命令が 32bit = 4 バイトなので, 次の命令の番地は 4 大きい

(注 1) 符号付き整数が 2 の補数表現なので, `addu` の演算結果は `add` 全く同じになる. `subu` の演算結果も `sub` と全く同じになる. 本講義では, 特にこれらの命令を区別しなくても良い. [4]

(注 2) `slt` は “set on less than” (「小なり」の場合にセットする (1 にする)) の略. `slt` と `sltu` は動作が異なる. `slt` は 2 数を符号付き整数と見て比較するのに対し, `sltu` は 2 数を符号無し整数と見て比較する.

練習 3.1 汎用レジスタと主記憶の値 (10 進数) が表の通りであるとする.

レジスタ	$5	$6	$7	$8
値	12	23	-16	512

下記の各命令を実行したときに, データの値が変化するレジスタと, その値を示せ.

0. `add $20,$6,$7`　(解答例) `$20 = $6+$7 = 23+(-16) = 7` なので, `$20 = 7`
1. `sub $21,$8,$5`
2. `or $22,$5,$0`
3. `slt $23,$5,$6`

3.1.3 レジスタと定数の演算

- 定数の加算命令

  `addi $21,$7,30` (add immediate 命令)

  – (動作) `$21` = $7 + 30 , PC = PC+4

  – 命令中に書かれた定数のことを 即値 (immediate value/data) と言う

  一般形は `addi` Rt, Rs, Imm (Rt, Rs はレジスタ, Imm は定数)

  – (動作) $Rt = Rs + sx(Imm)$, PC = PC + 4

- 即値データ (Imm) は 32bit より短い → 32bit に拡張する必要がある

  – 機械語では命令全体を 32bit で表すため, 命令に含まれる定数はどうしても 32bit よりも短くなる

  – Imm を 32bit に { 符号拡張 (符号付き整数の演算の場合) / ゼロ拡張 (符号無し整数の演算や論理演算の場合) }

- レジスタと定数の演算命令

命令	動作	説明
`addi` Rt,Rs,Imm	$Rt = Rs + sx(Imm)$, PC = PC + 4	加算 (符号付き)
`addiu` Rt,Rs,Imm	$Rt = Rs + zx(Imm)$, PC = PC + 4	加算 (符号無し)
`andi` Rt,Rs,Imm	$Rt = Rs \,\&\, zx(Imm)$, PC = PC + 4	ビット毎の論理積
`ori` Rt,Rs,Imm	$Rt = Rs \mid zx(Imm)$, PC = PC + 4	ビット毎の論理和
`xori` Rt,Rs,Imm	$Rt = Rs \oplus zx(Imm)$, PC = PC + 4	ビット毎の排他的論理和
`nori` Rt,Rs,Imm	$Rt = \tilde{}(Rs \mid zx(Imm))$, PC = PC + 4	ビット毎の論理和否定
`slti` Rt,Rs,Imm	$Rt = (Rs < sx(Imm))$? 1 : 0, PC = PC + 4	比較「小なり」 (符号付き)
`sltiu` Rt,Rs,Imm	$Rt = (Rs < zx(Imm))$? 1 : 0, PC = PC + 4	比較「小なり」 (符号無し)
`sll` Rd,Rs,Sa	$Rd = Rs <<_L Sa$, PC = PC + 4	論理左シフト
`srl` Rd,Rs,Sa	$Rd = Rs >>_L Sa$, PC = PC + 4	論理右シフト
`sra` Rd,Rs,Sa	$Rd = Rs >>_A Sa$, PC = PC + 4	算術右シフト

☆ 算術左 シフト命令は存在しない. 左シフトはすべて論理左シフトで行う (符号ビットは無視).

[4] 参考までに, これらの命令の違いは, オーバーフローが起こったときに, `add` と `sub` では割り込みが発生するが, `addu` と `subu` では発生しないことである.

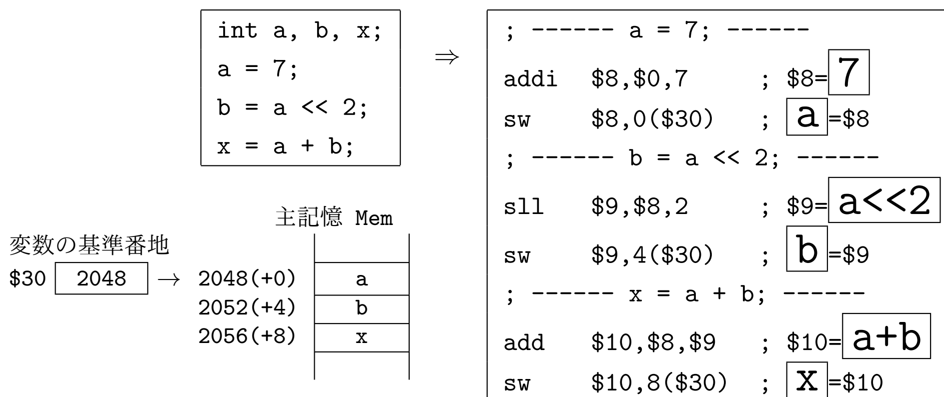
3.1.4 主記憶とのデータ転送命令

- ロード命令 (主記憶からレジスタへのデータ転送命令)
  `lw $10,12($7)` (load word 命令)
  - (動作) $10 = Mem[$7+12,4] , PC = PC + 4
    - ∗ 主記憶の $7+12 番地から始まる 4 バイト (1 ワード) データを, レジスタ $10 に転送 (コピー) する
    - ∗ $7=1024 のときに lw $10,12($7) を実行 → $10=Mem[1036 ,4] となる
    - ∗ $7=1024 のときに lw $10,-12($7) を実行 → $10=Mem[1012 ,4] となる (負の数も書ける)
- ストア命令 (レジスタから主記憶へのデータ転送命令) の例
  `sw $10,12($7)` (store word)
  - (動作) Mem[$7+12,4] = $10 , PC = PC + 4
- 主記憶とのデータ転送命令 (一般形)

lw $Rt, Imm(Rs)$	Rt = Mem[$Rs + sx(Imm)$, 4], PC = PC + 4	主記憶→レジスタ (32bit)
sw $Rt, Imm(Rs)$	Mem[$Rs + sx(Imm)$, 4] = Rt, PC = PC + 4	レジスタ→主記憶 (32bit)

3.1.5 MIPS の演算命令の使われ方

C 言語の計算式や代入文は, 例えば次のようにコンパイルされる



練習 3.2 汎用レジスタと主記憶の値 (10 進数) が表の通りであるとする.

レジスタ	$5	$6	$7	$8
値	12	23	-16	512

主記憶の番地	512〜515	516〜519	520〜523	524〜527
値	123	222	753	-77

下記の各命令を実行したときに, データの値が変化するレジスタと, その値を示せ.

1. addi $31,$5,-20
2. sra $7,$7,2
3. lw $6,12($8)

3.2 アセンブリと機械語

命令はコンピュータ内では全て ビット列 で表現される

– (例) `addi $11,$5,23` ⇒

6 bit	5 bit	5 bit	16 bit
001000	00101	01011	0000000000010111
addi	5	11	23

= 0010 0000 1010 1011 0000 0000 0001 0111 = x 20AB0017

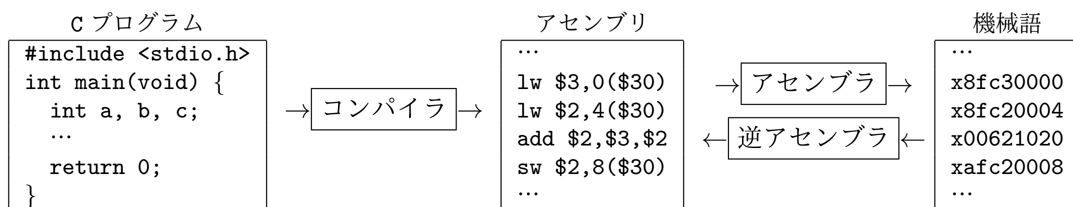
– アセンブリプログラム (assembly program)

addi, $11, 23 等の 記号表記 (ニモニック[5]) によるプログラム
これを「アセンブラ」「アセンブラプログラム」と呼ぶ人もいる

– 機械語 (machine language)

∗ ビット列によるプログラム (コンピュータが直接実行できる)

– アセンブラ (assembler) ··· アセンブリを機械語に 変換 する処理系 (プログラム)



3.3 MIPS の機械語

3.3.1 MIPS の命令フォーマット

- MIPS の命令は全て 32 bit (4 バイト) で符号化される
- 命令フォーマット (instruction format ; 機械語の パターン) は 3 種類

(1) R-type

6bit	5bit	5bit	5bit	5bit	6bit
XXXXXX	*Rs*	*Rt*	*Rd*	*Sa*	XXXXXX

(2) I-type

6bit	5bit	5bit	16bit
XXXXXX	*Rs*	*Rt*	*Imm*

(3) J-type

6bit	26bit
XXXXXX	*Addr*

Rd, *Rs*, *Rt* ··· 汎用レジスタの番号 (0∼31)
Imm ··· 定数 (16 ビット),　*Addr* ··· 定数 (26 ビット),　*Sa* ··· 定数 (5 ビット)
XXXXXX ··· 命令毎に異なるビット列が入る

- フィールド (field)

機械語命令の 区切られた 各部分 (001000, 00101, 01011 等)

[5]英語は mnemonic で「記憶を助けるもの」の意

3.3.2 MIPS の機械語

(1) R-type

ニモニック	機械語						動作
	6bit	5bit	5bit	5bit	5bit	6bit	
add *Rd*,*Rs*,*Rt*	000000	*Rs*	*Rt*	*Rd*	00000	100000	*Rd*=*Rs*+*Rt*, PC=PC+4
addu *Rd*,*Rs*,*Rt*	000000	*Rs*	*Rt*	*Rd*	00000	100001	*Rd*=*Rs*+*Rt*, PC=PC+4
sub *Rd*,*Rs*,*Rt*	000000	*Rs*	*Rt*	*Rd*	00000	100010	*Rd*=*Rs*-*Rt*, PC=PC+4
subu *Rd*,*Rs*,*Rt*	000000	*Rs*	*Rt*	*Rd*	00000	100011	*Rd*=*Rs*-*Rt*, PC=PC+4
and *Rd*,*Rs*,*Rt*	000000	*Rs*	*Rt*	*Rd*	00000	100100	*Rd*=*Rs*&*Rt*, PC=PC+4
or *Rd*,*Rs*,*Rt*	000000	*Rs*	*Rt*	*Rd*	00000	100101	*Rd*=*Rs* | *Rt*, PC=PC+4
xor *Rd*,*Rs*,*Rt*	000000	*Rs*	*Rt*	*Rd*	00000	100110	*Rd*=*Rs*$\oplus$*Rt*, PC=PC+4
nor *Rd*,*Rs*,*Rt*	000000	*Rs*	*Rt*	*Rd*	00000	100111	*Rd*=~(*Rs* | *Rt*), PC=PC+4
sll *Rd*,*Rt*,*Sa*	000000	00000	*Rt*	*Rd*	*Sa*	000000	*Rd*=*Rt*<<$_L$ *Sa*, PC=PC+4
srl *Rd*,*Rt*,*Sa*	000000	00000	*Rt*	*Rd*	*Sa*	000010	*Rd*=*Rt*>>$_L$ *Sa*, PC=PC+4
sra *Rd*,*Rt*,*Sa*	000000	00000	*Rt*	*Rd*	*Sa*	000011	*Rd*=*Rt*>>$_A$ *Sa*, PC=PC+4
slt *Rd*,*Rs*,*Rt*	000000	*Rs*	*Rt*	*Rd*	00000	101010	*Rd*=(*Rs*<*Rt*)?1:0 (符号付き比較), PC=PC+4
sltu *Rd*,*Rs*,*Rt*	000000	*Rs*	*Rt*	*Rd*	00000	101011	*Rd*=(*Rs*<*Rt*)?1:0 (符号無し比較), PC=PC+4

(2) I-type

ニモニック	機械語				動作
	6bit	5bit	5bit	16bit	
addi *Rt*,*Rs*,*Imm*	001000	*Rs*	*Rt*	*Imm*	*Rt*=*Rs*+*sx*(*Imm*), PC=PC+4
addiu *Rt*,*Rs*,*Imm*	001001	*Rs*	*Rt*	*Imm*	*Rt*=*Rs*+*sx*(*Imm*), PC=PC+4
andi *Rt*,*Rs*,*Imm*	001100	*Rs*	*Rt*	*Imm*	*Rt*=*Rs*&*zx*(*Imm*), PC=PC+4
ori *Rt*,*Rs*,*Imm*	001101	*Rs*	*Rt*	*Imm*	*Rt*=*Rs* | *zx*(*Imm*), PC=PC+4
xori *Rt*,*Rs*,*Imm*	001110	*Rs*	*Rt*	*Imm*	*Rt*=*Rs*$\oplus$*zx*(*Imm*), PC=PC+4
slti *Rt*,*Rs*,*Imm*	001010	*Rs*	*Rt*	*Imm*	*Rt*=(*Rs*<*sx*(*Imm*))?1:0 (符号付き比較), PC=PC+4
sltiu *Rt*,*Rs*,*Imm*	001011	*Rs*	*Rt*	*Imm*	*Rt*=(*Rs*<*zx*(*Imm*))?1:0 (符号無し比較), PC=PC+4
lw *Rt*,*Imm*(*Rs*)	100011	*Rs*	*Rt*	*Imm*	*Rt*=Mem[*Rs*+*sx*(*Imm*),4], PC=PC+4
sw *Rt*,*Imm*(*Rs*)	101011	*Rs*	*Rt*	*Imm*	Mem[*Rs*+*sx*(*Imm*),4]=*Rt*, PC=PC+4
beq *Rt*,*Rs*,*Imm*	000100	*Rs*	*Rt*	*Imm*	PC=(*Rt*==*Rs*) ? (PC+4)+*sx*(*Imm*)*4 : (PC+4)
bne *Rt*,*Rs*,*Imm*	000101	*Rs*	*Rt*	*Imm*	PC=(*Rt*!=*Rs*) ? (PC+4)+*sx*(*Imm*)*4 : (PC+4)

(3) J-type

ニモニック	機械語		動作
	6bit	26bit	
j *Addr*	000010	*Addr*	PC=PC$_{31:28}$ || *Addr**4

例題 3.1 addi $13,$5,-3 の機械語を (16 進表示で) 示せ.

I-type の addi の行 → | addi *Rt*,*Rs*,*Imm* | 001000 | *Rs* | *Rt* | *Imm* | *Rt*=*Rs*+*sx*(*Imm*), PC=PC+4 |

addi $13,$5,-3

⇒ | 001000 | $5 | $13 | -3 | (レジスタの順序に注意)

⇒ | 001000 | 00101 | 01101 | 1111 1111 1111 1101 |

⇒ 0010 0000 1010 1101 1111 1111 1111 1101

⇒ x 20ADFFFD

☆ sll $0,$0,0 を NOP として用いることが多い. それは, 機械語が x 00000000 だから.

例題 3.2 機械語が x012A4026 である命令のアセンブリ表記を示せ.

x012A4026

⇒ 0000 0001 0010 1010 0100 0000 0010 0110 (→ まず先頭 6bit を見る)

⇒ 000000 01001010100100000000100110 (先頭 6bit が 000000 → 末尾 6bit を見る)

⇒ 000000 01001010100100000000 100110 (末尾 6bit が 100110 → xor とわかる)

⇒ 000000 01001 01010 01000 00000 100110

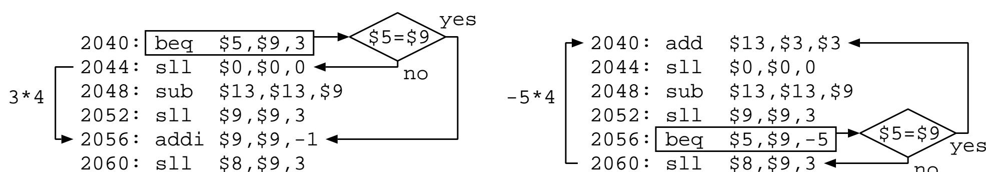
⇒ 000000 $9 $10 $8 00000 100110 ⇒ xor $8,$9,$10

3.4 MIPS の分岐命令

- 条件分岐命令 (等しいときに分岐)

  `beq $5,$9,3` (branch if equal)

  - (動作) PC = ($5==$9) ? (PC + 4) + 3*4 : (PC + 4)
    - ∗ 5 番レジスタと 9 番レジスタの値を比較
    - ∗ 等しければ, 分岐 (次の命令の 3 個先の命令 にジャンプ)
    - ∗ 等しくなければ, 次の命令を実行



一般形は `beq` *Rt*, *Rs*, *Imm*

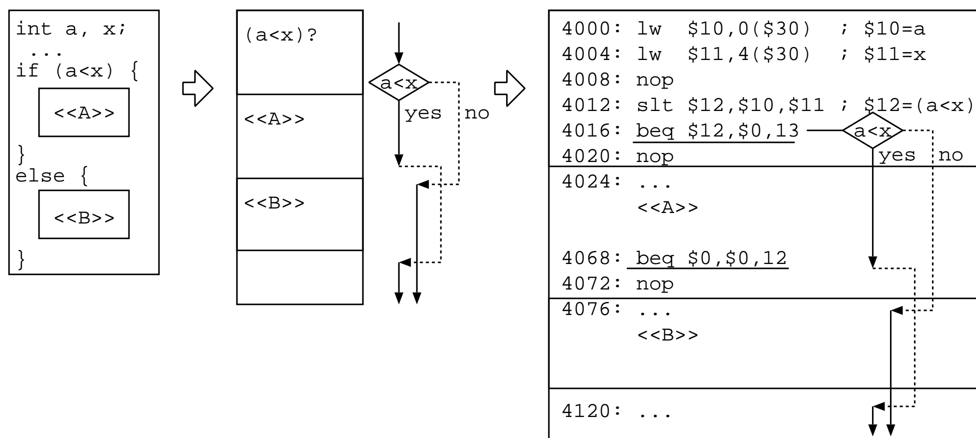
  - (動作) PC = (*Rt* == *Rs*) ? (PC + 4) + *sx*(*Imm*)*4 : (PC + 4)

- 条件分岐命令 (等しくないときに分岐)

  `bne` *Rt*, *Rs*, *Imm* (branch if not equal)

  - (動作) PC = (*Rt* != *Rs*) ? (PC + 4) + *sx*(*Imm*)*4 : (PC + 4)

- C 言語の if 文は例えば次のようにコンパイルされる
  - a の番地は $30+0, x の番地は $30+4 であるとする
  - 4008, 4020, 4072 の nop が必要な理由は 10 章で述べる (少し難解)



☆ これまでに紹介したのは, MIPS の整数演算命令のうちの主要なものである

  - MIPS はこの他に浮動小数点演算命令も備えている.
  - MIPS には R2000, R3000, R4000 等多くの系列 CPU があり, 命令セットが拡張されている.

練習の解答例

練習 3.1

1. `$21 = $8-$5 = 512-12 = 500`. よって `$21 = 500`.
2. `$22 = $5|$0 = 12|0 = 12`. よって `$22 = 12`.
3. `$23 = ($5<$6)?1:0 = (12<23)?1:0 = 1`. よって `$23 = 1`.

練習 3.2

1. `$31 = $5+(-20) = 12+(-20) = -8`. よって `$31 = -8`.
2. `$7 = $7>>`$_A$`2 = -16>>`$_A$`2 = -16/4 = -4`. よって `$7 = -4`.
3. `$6 = Mem[$8+12,4] = Mem[512+12,4] = Mem[524,4] = -77`. よって `$6 = -77`.

Nagisa ISHIURA