計算機科学実験及演習 3
SIMPLE 設計資料

ver 1.0: 1995.1.18

情報工学教室：SIMPLE 担当グループ

1 はじめに

本資料は，計算機科学実験及演習 3A/B の課題の一つである，コンピュータのハードウェア設計に関するものである。設計するコンピュータ SIMPLE (SIxteen-bit MicroProcessor for Laboratory Experiment) は，1 語が 16 ビットで命令セットも簡単なものだが，コンピュータが持つべき基本機能は一通り備えられている。従って SIMPLE の設計を正しく行なうことが，より高度なコンピュータへ進む大きな第一歩となるだろう。

本資料では，まず SIMPLE の基本的なアーキテクチャ，即ちどのようなレジスタやメモリがあって，どのような形式と機能を持つ命令が実行されるかを示す。次に，このアーキテクチャに基づき，最も単純に設計する方法のごく一部を示す。

また，単に与えられた仕様と方針に沿って設計するだけではなく，学生諸君が独自の改良を施すことが強く期待されている。そこで学生諸君の参考となるように，アーキテクチャやハードウェア回路の改良の例を最後に示す。

2 SIMPLE のアーキテクチャ

レジスタやメモリなどコンピュータが持つ様々な資源と，それを操作する命令の機能を定義したものを，**アーキテクチャ**と呼ぶ。アーキテクチャは原則として論理的な構造や機能を定めるもので，それをどのように実現するかを定義するものではない。従って一つのアーキテクチャに基づいて，色々なハードウェアが設計されるのが普通である。但しどのような設計を行なっても，プログラムの動作が同じであることが重要なポイントである。

2.1 主記憶とレジスタ

SIMPLE は 1 語 16 ビットのコンピュータであり，主記憶やレジスタはいずれも 16 ビット幅である。

- 主記憶
  主記憶のアドレスは 16 ビットであり，語単位にアドレスが付けられる。従ってアドレス空間の大きさは 64 KW である。アドレスが `a` の語のアクセスは，(C 言語風に) `*(a)` と表記する。
- 汎用レジスタ
  8 個の汎用レジスタが備えられ，`r[0]`, `r[1]`, ..., `r[7]` と表記される。演算のソース／デスティネーションや主記憶のアドレス計算に用いられる。
- プログラム・カウンタ
  実行中の命令のアドレスを保持する。`PC` と表記する。
- 条件コード
  演算命令の結果に基づく分岐条件を保持する 3 個のフラグ `S`, `Z`, `C` からなり，以下のように設定される。
  `S`: 負ならば 1，そうでなければ 0
  `Z`: ゼロならば 1，そうでなければ 0
  `C`: 桁上げがあれば 1，そうでなければ 0
  なお，加減算と比較命令以外での `C` の値は後に述べる。

- 入出力フラグ

  入出力機器の状態を示す 2 個のフラグ STI と STO からなり，以下の意味を持つ。

  STI : 入力操作が可能であれば 1，そうでなければ 0

  STO : 出力操作が可能であれば 1，そうでなければ 0

2.2 命令セット・アーキテクチャ

命 令 形 式

SIMPLE の命令は全て 1 語（16 ビット）の固定長であり，図 1 に示すように 4 種類の命令形式がある。各命令形式とフィールドの意味は以下の通りである。

(a) ロード／ストア命令形式

- $I_{15:14}$ (op1) ..... 操作コード (00/01)
- $I_{13:11}$ (Ra) ..... ソース／デスティネーションのレジスタ番号
- $I_{10:8}$ (Rb) ..... ベース・レジスタ番号
- $I_{7:0}$ (d) ..... 変位

(b) 即値ロード／無条件分岐命令形式

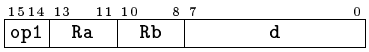
- $I_{15:14}$ (op1) ..... 操作コード (10)
- $I_{13:11}$ (op2) ..... 操作コード (000 ～ 110)
- $I_{10:8}$ (Rb) ..... ソース／デスティネーション／ベースのレジスタ番号
- $I_{7:0}$ (d) ..... 即値または変位

(c) 条件分岐命令形式

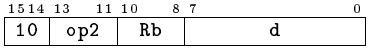
- $I_{15:14}$ (op1) ..... 操作コード (10)
- $I_{13:11}$ (op2) ..... 操作コード (111)
- $I_{10:8}$ (cond) .... 分岐条件
- $I_{7:0}$ (d) ..... 変位

(d) 演算／制御命令形式

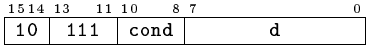
- $I_{15:14}$ (op1) ..... 操作コード (11)
- $I_{13:11}$ (Rs) ..... ソース・レジスタ番号
- $I_{10:8}$ (Rd) ..... デスティネーション・レジスタ番号
- $I_{7:4}$ (op3) ..... 操作コード (0000 ～ 1111)
- $I_{3:0}$ (d) ..... シフト桁数／ IN 命令修飾



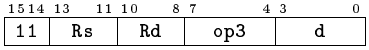
(a) ロード／ストア命令形式



(b) 即値ロード／無条件分岐命令形式



(c) 条件分岐命令形式



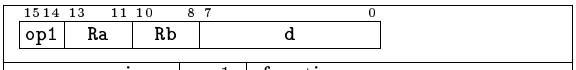
(d) 演算／制御命令形式

図 1　SIMPLE の命令形式

ロード／ストア命令

SIMPLE のロード命令 LD (load) とストア命令 ST (store) の機能を表 1 に示す。ソース／デスティネーションは，フィールド Ra で指定されたレジスタ r[Ra] である。また実効アドレスはベース・レジスタ・アドレス指定により，フィールド Rb で指定されたレジスタ r[Rb] と，フィールド d を符合拡張した sext(d) を加算して求める。

表 1　SIMPLE のロード／ストア命令



mnemonic	op1	function
LD Ra,d(Rb)	00	r[Ra] = *(r[Rb] + sext(d))
ST Ra,d(Rb)	01	*(r[Rb] + sext(d)) = r[Ra]

即値ロード／無条件分岐命令

SIMPLE の即値ロード命令 LI (load immediate) と，三つの無条件分岐命令 BL (branch long), BAL (branch and link), B (branch) の機能を表 2 に示す。

- LI .... 即値 sext(d) をレジスタ r[Rb] に格納する。
- BL .... ベース・レジスタ・アドレス指定で求めたアドレス r[Rb] + sext(d) に分岐する。例えば復帰アドレスが r[1] に保持されているようなサブルーチンからは，BL 0(R1) によって復帰することができる。
- BAL ... 次の命令のアドレス PC + 1 を復帰アドレスとして r[Rb] に格納するとともに，PC 相対アドレス指定による分岐を行なう。なお分岐アドレスは（PC ではなく）PC + 1 に sext(d) を加算して求める。
- B ..... Rb と d を連結した conc(Rb,d) を符合拡張した値を変位として，PC 相対アドレス指定による分岐を行なう。

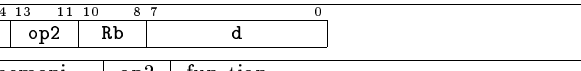
なお “(reserved)” と記された命令は何の動作もせず，単に次の命令に移行する。

条件分岐命令

SIMPLE の条件分岐命令は表 3 に示すように，フィールド cond で定められる分岐条件が成り立てば PC 相対アドレスによる分岐を行ない，成り立たなければ単に次の命令に移行する。各命令の分岐条件は以下の通り。

- BE (branch on equal-to) ..... 条件コード Z が 1
- BLT (branch on less-than) ..... 条件コード S が 1
- BLE (branch on less-than or equal-to) ..... Z または S が 1
- BC (branch on carry) ..... 条件コード C が 1
- BNE (branch on not-equal-to) ..... Z が 0

表 2　SIMPLE の即値ロード／無条件分岐命令



mnemonic	op2	function
(reserved)	000	
LI Rb,d	001	r[Rb] = sext(d)
(reserved)	010	
(reserved)	011	
BL d(Rb)	100	PC = r[Rb] + sext(d)
BAL Rb,d	101	r[Rb] = PC + 1 ; PC = PC + 1 + sext(d)
B d	110	PC = PC + 1 + sext(conc(Rb,d))

表 3　SIMPLE の条件分岐命令

15 14	13 11	10 8	7 0
10	111	cond	d

mnemonic	cond	function
BE d	000	if (Z) PC = PC + 1 + sext(d)
BLT d	001	if (S) PC = PC + 1 + sext(d)
BLE d	010	if (Z || S) PC = PC + 1 + sext(d)
BC d	011	if (C) PC = PC + 1 + sext(d)
BNE d	100	if (!Z) PC = PC + 1 + sext(d)
BGE d	101	if (!S) PC = PC + 1 + sext(d)
BGT d	110	if (!(Z || S)) PC = PC + 1 + sext(d)
BNC d	111	if (!C) PC = PC + 1 + sext(d)

- BLT (branch on greater-than or equal-to) ... S が 0
- BLE (branch on greater-than) ..... Z および S が 0
- BC (branch on not-carry) ..... C が 0

演算／制御命令

SIMPLE の演算／制御命令を表 4 に示す。演算命令では結果に基づく条件コードが設定される。

(1) 算術演算

レジスタ r[Rd] と r[Rs] の加算 (ADD, add) または減算 (SUB, subtract) の結果を r[Rd] に格納し，条件コードを設定する。C には最上位ビットからの桁上げが設定される。

(2) 論理演算

レジスタ r[Rd] と r[Rs] の，ビットごとの論理積 (AND, and), 論理和 (OR, or)，または排他的論理和 (XOR, exclusive-or) の結果を r[Rd] に格納し，条件コードを設定する。但し C は演算結果に関わらず 0 となる。

(3) 比較演算 (CMP, compare)

レジスタ r[Rd] から r[Rs] を減算し，結果に基づく条件コード設定のみを行なう。C には最上位ビットからの桁上げが設定される。

(4) 移動演算 (MOV, move)

レジスタ r[Rd] に r[Rs] の値を単に格納し，r[Rs] の値に基づき条件コードを設定する。但し C は r[Rs] に関わらず 0 となる。

(5) シフト演算

レジスタ r[Rd] の値を，左論理シフト (SLL, shift left logical)，左循環シフト (SLR, shift left rotate)，右論理シフト (SRL, shift right logical)，または右算術シフト (SRA, shift right arithmetic) した値を r[Rd] に格納し，条件コードを設定する。シフト桁数は d の即値 (0 ～ 15) である。また C には，シフト桁数が 0 の時または SLR では 0 が，それ以外では最後にシフト・アウトされたビットの値が設定される。

(6) 制御命令

- IN (input) ..... d の最下位ビットが 0 の時は，(スイッチなどの) 機器から入力した値をレジスタ r[Rd] に格納し，入出力フラグ STI を 0 にする。d の最下位ビットが 1 の時は，r[Rd] のビット 0 とビット 1 に入出力フラグ STI と STO の値を格納し，その他のビットは 0 とする。
- OUT (output) ... レジスタ r[Rs] の値を (16 進数表示 LED などの) 機器に出力し，入出力フラグ STO を 0 にする。
- HLT (halt) ..... SIMPLE を停止させる。

3　基本的な設計

表 4　SIMPLE の演算／制御命令

15 14	13 11	10 8	7 4	3 0
11	Rs	Rd	op3	d

mnemonic	op3	function
ADD Rd,Rs	0000	r[Rd] = r[Rd] + r[Rs]
SUB Rd,Rs	0001	r[Rd] = r[Rd] - r[Rs]
AND Rd,Rs	0010	r[Rd] = r[Rd] & r[Rs]
OR Rd,Rs	0011	r[Rd] = r[Rd] | r[Rs]
XOR Rd,Rs	0100	r[Rd] = r[Rd] ^ r[Rs]
CMP Rd,Rs	0101	r[Rd] - r[Rs]
MOV Rd,Rs	0110	r[Rd] = r[Rs]
(reserved)	0111	
SLL Rd,d	1000	r[Rd] = shift_left_logical(r[Rd],d)
SLR Rd,d	1001	r[Rd] = shift_left_rotate(r[Rd],d)
SRL Rd,d	1010	r[Rd] = shift_right_logical(r[Rd],d)
SRA Rd,d	1011	r[Rd] = shift_right_arithmetic(r[Rd],d)
IN Rd,d	1100	r[Rd] = (d & 1) ? io_status : input)
OUT Rs	1101	output = r[Rs]
(reserved)	1110	
HLT	1111	halt()

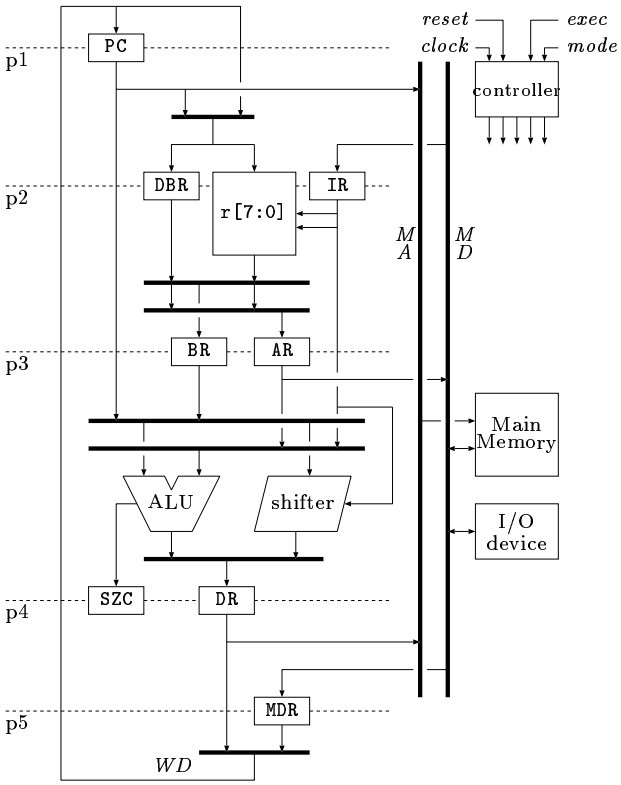


図 2　SIMPLE/B のブロック図

前述のように，一つのアーキテクチャに基づいて様々なハードウェアを設計できる。本節では SIMPLE の CPU を 5 つのフェーズに分割したハードウェア SIMPLE/B を，基本的な設計例として示す。なお，後にに示すような改良の余地がいくつもあり，SIMPLE/B に因われることなく，学生諸君自身充分に学んだ後で読者自信の創意により設計することを強く期待する。

SIMPLE/B は図 2 に示すように，順次活性化する p1 ～ p5 の 5 つのフェーズ，各フェーズに制御信号を供給する制御回路，主記憶，及びスイッチと 16 進数表示 LED からなる入出力機器から構成される。なお図中の太線はバスまたはセレクタを表す。

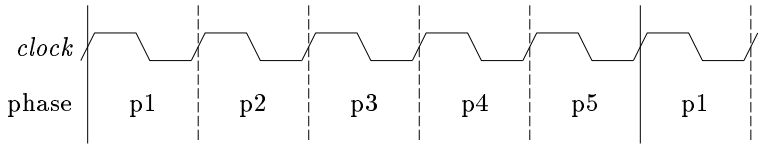


図 3　SIMPLE/B のフェーズ

3.1　制 御 回 路

SIMPLE/B には以下の信号が外部から供給される。

(1) *clock*

動作クロック。適切な発振回路を用いて，Hi/Lo の期間ができるだけ 1 : 1 に近いクロックを供給する。制御回路は図 3 に示すように，クロックの立ち上がりエッジに同期して各フェーズを一つずつ順番に活性化する。従って，クロックの周期は遅延が最大であるようなフェーズの遅延時間により定まり，多くの場合は主記憶のアクセス時間に周辺回路の遅延時間などを加えた値となる。

(2) *reset*

リセット信号。プッシュ・スイッチを用いて，スイッチを押すと 1 が，離すと 0 が供給されるようにする†。また電源投入時にも 1 になるようにするのが望ましい。*reset* が 1 になると SIMPLE/B は適切な初期状態に移行する。

(3) *exec*

起動／停止信号。プッシュ・スイッチを用いて，スイッチを押すと 1 が，離すと 0 が供給されるようにする†。*exec* が 0 から 1 に変化すると，SIMPLE/B が実行状態にあればその時点で実行中の命令を完了してから停止する。停止状態にあれば *mode* によって定まる動作をする。

(4) $mode_{1:0}$

実行モード。SIMPLE/B が停止状態の時に *exec* が 0 から 1 に変化すると，トグル・スイッチで設定された *mode* の値に従って以下の動作を行なう。

- $mode = 00$ .... 実行開始。
- $mode = 01$ .... デバッグ用命令のセット。入力スイッチの値を命令レジスタ (IR) にセットする。
- $mode = 10$ .... ステップ実行。PC が保持するアドレスの命令を実行して停止する。なお直前にデバッグ用命令がセットされていれば，IR が保持する命令を実行し，分岐が行なわれた時を除いて PC を更新しない。
- $mode = 11$ .... デバッグ用ステップ実行。ステップ実行と同様であるが，汎用レジスタの代わりにデバッグ用レジスタ (DBR) を命令のソース／デスティネーションとして使用する。

3.2　p1 ～ p5

SIMPLE/B は一つの命令を，$p1 \sim p5$ の 5 つのフェーズを順番に活性化しながら実行する。

(1) p1（命令フェッチ）

PC が保持するアドレスの命令を主記憶からフェッチし，IR に格納するとともに，PC に 1 を加える。

(2) p2（レジスタ読出）

IR が保持する命令の，Ra/ Rs フィールドと Rb/ Rd フィールドで指定される汎用レジスタの値を読出し，それぞれレジスタ AR と BR に格納する。但しデバッグ用ステップ実行の際には，DBR の値を AR と BR に

† スイッチを押すと 0 になるような$\overline{reset}$や$\overline{exec}$を供給しても構わない。

格納する。

(3) p3（演算）

ALU またはシフト回路により，命令が定める演算（アドレス計算や単なるデータ移動を含む）を行ない，その結果をレジスタ DR に格納する。また演算命令では条件コード S, Z, C をセットする。演算のソースには AR, BR の他，PC や命令の即値が用いられることがある。

(4) p4（主記憶アクセス）

ロード／ストア命令では，DR の値をアドレスとして主記憶をアクセスする。ロード命令では読出した値をレジスタ MDR に格納し，ストア命令では AR が保持する値を書込む。また入出力命令では，入力スイッチの値の MDR への格納や，AR の値の LED への表示を行う。

(5) p5（レジスタ書込）

汎用レジスタの書込を伴う命令では，DR，MDR または PC の値を，命令の Ra または Rb/ Rs フィールドで指定される汎用レジスタ（デバッグ用ステップ実行の際は DBR）に書込む。また分岐命令では DR の値を分岐先アドレスとして PC に書込む。

3.3 主記憶と入出力機器

主記憶の容量は SIMPLE のアドレス空間の大きさと同じ 64 KW とし，前半の 32 KW を (E)EPROM で構成された ROM 領域，後半の 32 KW を SRAM で構成された RAM 領域とする。またアドレス変換などは行なわず，命令で定められるアドレスをそのまま主記憶のアドレスとする。

入力機器は 16 個のトグル・スイッチと，その値を SIMPLE/B のクロックに同期させるためのレジスタからなる。また出力機器は出力すべき値を保持するレジスタと，その値を表示する 16 進数表示 LED からなる。また STI と STO の値を保持するフリップ・フロップ，それらを 1 にセットするためのプッシュ・スイッチ，及びそれらの値を表示するための LED も備える。

なお主記憶のアドレスは，PC または DR の値を選択した *MA* である。また主記憶と入出力機器の読出／書込データは，バス *MD* の値である。

4 アーキテクチャの改良

2 節で述べた SIMPLE の基本アーキテクチャには，様々な改良の余地がある。本節では，学生諸君の参考のために，以下の項目に関する拡張アーキテクチャの例を示す。

(1) 即値オペランドの強化

(2) 入出力命令の強化

(3) 割込のサポート

(4) アドレス空間の拡張

なお，これらの項目の独立性は比較的高いので，一部のみを採用することも可能である。また，ここで示すアーキテクチャはあくまで一例であるので，学生諸君が独自の改良を行なって構わないし，またそれを期待している。

4.1 即値オペランドの強化

基本アーキテクチャでは，演算命令のオペランドはいずれも汎用レジスタであり，例えば r[0] に 1 を加えるには

```
LI   R1, 1
ADD  R0, R1
```

表 5　演算命令の拡張と即値ロード命令の追加

15 14	13 11	10 8	7 0
10	op2	Rb	d

mnemonic	op2	function
LIL Rb,d	010	r[Rb] = (r[Rb] & 0xFF00) | d
LIH Rb,d	011	r[Rb] = (r[Rb] & 0x00FF) | (d << 8)

15 14	13 11	10 8	7 4	3	2 0
11	Rs	Rd	op3	i	d

mnemonic	op3	function
ADD Rd,Rs/immd	0000	r[Rd] = r[Rd] + opd(i,Rs,d)
SUB Rd,Rs/immd	0001	r[Rd] = r[Rd] - opd(i,Rs,d)
AND Rd,Rs/immd	0010	r[Rd] = r[Rd] & opd(i,Rs,d)
OR Rd,Rs/immd	0011	r[Rd] = r[Rd] | opd(i,Rs,d)
XOR Rd,Rs/immd	0100	r[Rd] = r[Rd] ^ opd(i,Rs,d)
CMP Rd,Rs/immd	0101	r[Rd] - opd(i,Rs,d)
MOV Rd,Rs/immd	0110	r[Rd] = opd(i,Rs,d)
(reserved)	0111	
SLL Rd,Rs/immd	1000	r[Rd] = shift_left_logical(r[Rd],opd(i,Rs,d))
SLR Rd,Rs/immd	1001	r[Rd] = shift_left_rotate(r[Rd],opd(i,Rs,d))
SRL Rd,Rs/immd	1010	r[Rd] = shift_right_logical(r[Rd],opd(i,Rs,d))
SRA Rd,Rs/immd	1011	r[Rd] = shift_right_arithmetic(r[Rd],opd(i,Rs,d))

表 6　入出力命令の拡張

15 14	13 11	10 8	7 4	3	2 0
11	Rs	Rd	op3	i	d

mnemonic	op3	function
IN Rd,Rs/immd	1100	r[Rd] = input[opd(i,Rs,d)]
OUT Rd,Rs/immd	1101	output[opd(i,Rs,d)] = r[Rd]

のように 2 命令を要する。そこで演算命令形式を以下のように変更し，r[Rs] だけではなく即値オペランドも指定できるようにする。

15 14	13 11	10 8	7 4	3	2 0
11	Rs	Rd	op3	i	d

- i = 0 .... operand = r[Rs]
- i = 1 .... operand = sext(conc(Rs,d))（Rs フィールドと d フィールドを結合したものを符合拡張した値）

このようにして得られるオペランドを opd(i,Rs,d) と表記すると，演算命令の仕様は表 5 に示すものとなる。なお，シフト命令のシフト桁数も opd(i,Rs,d) となることに注意せよ。

この他，即値ロード／無条件分岐命令形式の中で “(reserved)” であった二つの命令を，同表に示す LIL (load immediate lower)，LIU (load immediate upper) とする。これらは r[Rb] の下位／上位 8 ビットのみを即値 d とするもので，任意の 16 ビット整数，例えば $\mathrm{A25B}_{(16)}$ の r[0] へのロードは

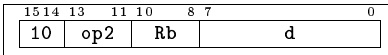
```
LIL  R0, 0x5B
LIH  R0, 0xA2
```

の 2 命令で実現できる。

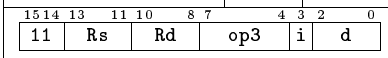
4.2　入出力命令の強化

基本アーキテクチャの入出力命令が操作できる入出力機器は，各々一つだけである。そこで複数の入出力機器を接続できるように，入出力機器にアドレスを与え，それを前述の opd(i,Rs,d) で指定するように変更する（表 6）。なお，OUT が出力するデータが r[Rs] から r[Rd] に変更されていることに注意せよ。

表 7　制御命令の拡張と TS 命令



mnemonic		op2	function
TS	d(Rb)	000	Z = (*(r[Rb]+sext(d))==0) ; *(r[Rb] + sext(d)) = 1



mnemonic		op3	function
TRAP	Rs/immd	0111	raise(opd(i,Rs,d))
LDSYS	Rd,sysreg	1110	r[Rd] = sysreg[conc(Rs,i,d)]
STSYS	Rd,sysreg	1111	sysreg[conc(Rs,i,d)] = r[Rd]

4.3　割込のサポート

割込をサポートするためには，割込が発生した時点でのコンテクスト保存と，割込処理の完了時の状態復元が必要である。そこで特殊なシステム・レジスタをいくつか用意し，それらを読み書きするために表 7 に示す LDSYS (load system register) と STSYS (store system register) 命令（基本アーキテクチャでは HLT）を追加する。またこれらの命令は通常のプログラムでは実行できない特権命令とし，特権／非特権を区別するためのカーネル／ユーザ・モードを用意する。なお，入出力命令も特権命令とするのが望ましい。

システム・レジスタにはアドレスが付けられ，LDSYS/STSYS の Rs, i, d を結合したもの（あるいはその一部）がアドレスとなる。どのようなシステム・レジスタを用意するかについては色々考えられるが，例えば以下のようなものがあげられる。

(1) HLT

疑似的なレジスタであり，STSYS を行なうと SIMPLE が停止する。

(2) RETI

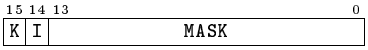
疑似的なレジスタであり，STSYS を行なうとシステム・レジスタ IPC が保持するアドレスへ分岐する。

(3) IPC

割込発生時の PC の値が自動的に保存され，RETI により復元される。

(4) IST

割込に関する種々の情報，例えば以下のようなものを保持する。



- K ..... 割込発生時のカーネル／ユーザ・モードが自動的に保存され，RETI により復元される。
- I ..... 1 であれば全ての割込が禁止される割込禁止ビット。割込発生時に自動的に 1 になり，RETI により 0 になる。
- MASK ..... 14 種類までの個々の割込について，対応するビットが 1 のものが禁止される。

(5) IWR0～ IWR3 汎用レジスタの保存／復元などに用いるための作業用レジスタ。

この他，特定の割込を起こすために追加される命令 TRAP (raise trap) は，識別番号が opd(i,Rs,d) の割込を発生する。

また割込に関連して TS (test and set) 命令も追加される。この命令は主記憶のある語の値が 0 か否かに基づいてて条件コード・フラグ Z をセットし，続いて同じ語に 1 を書込む。この主記憶の読出と書込を一つの命令で行なう機能は，複数のプロセス間で情報をやりとりするために必須のものである。

4.4　アドレス空間の拡張

SIMPLE のアドレス空間は 64KW であるが，大容量の RAM を使えばより大きな物理アドレス空間を実現

するのはさほど難しくない。また，割込の導入によって複数のプロセスを並行実行する多重プログラミングが可能になると，プロセスごとにアドレス空間を与える多重アドレス空間があると便利である。そこで，アドレス空間の簡単な拡張法として，セグメント・レジスタを用いる方法を示す。

プロセスには物理アドレス空間中の連続した領域（セグメント）が与えられ，セグメント・レジスタがその先頭アドレスを保持する。従ってロード／ストア命令がアクセスする物理アドレスは，実効アドレス（`r[Rb]+sext(d)`）にセグメント・レジスタの値を加えたものとなる。

セグメント・レジスタの設定はオペレーティング・システムなどの仕事であるので，セグメント・レジスタはシステム・レジスタの一つとすることができる。また，実効アドレスの上限／下限を保持するシステム・レジスタと自動的なチェック機構，命令コード領域とデータ領域の分離など，様々な機能追加が考えられる。

5 ハードウェアの改良

SIMPLE/B の改良法の例として，フェーズの並列実行による命令サイクルの短縮をあげる。

(1) p1/p5 の並列実行（図 4(a)）

命令サイクルは 4 サイクルになる。分岐先アドレスを PC へセットする操作を少し工夫すれば，簡単に実現できる。

(2) p1/p3 と p2/p5 の並列実行（図 4(b)）

命令サイクルは 3 サイクルになる。

p1/p3 を常に並列実行するためには，分岐命令の実行をかなり工夫する必要がある（ヒント：分岐アドレス計算に p3 を使わない）。別の方法として分岐命令だけは並列実行しないというのも考えられる。

p2/p5 の並列実行については，汎用レジスタを更新する命令の直後に，同じレジスタを参照する命令が現れた時の処置を工夫しなければならない。

(3) p1/p3/p5 と p2/p4 の並列実行（図 4(c)）

命令サイクルは 2 サイクルになる。(2) のデータ依存の解決を更に工夫しなければならない。

(4) p1/p2/p3/p4/p5 の並列実行（図 4(d)）

いわゆるパイプラインであり，命令サイクルは 1 サイクルになる。実現は飛躍的に困難になるが，問題点の抽出や解決法の検討だけでも行なって欲しい。

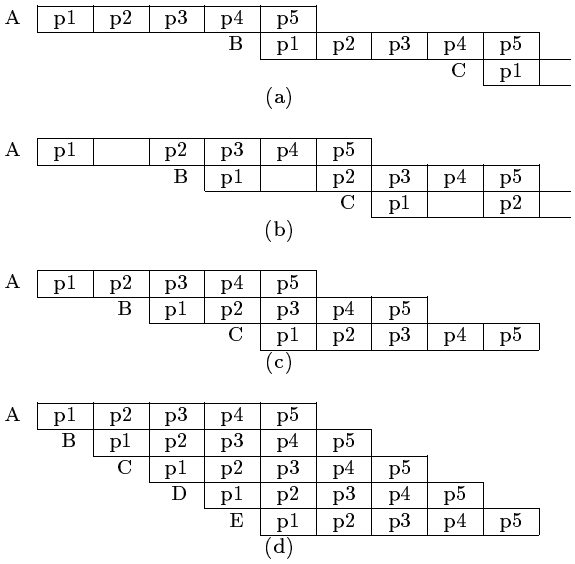
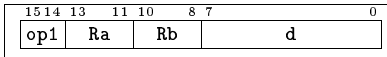
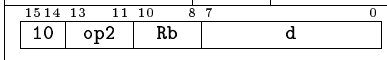


図 4　フェーズの並列実行

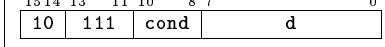
付録 A　基本命令セット



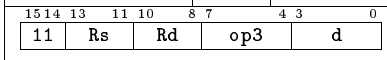
mnemonic	op1	function
LD Ra,d(Rb)	00	r[Ra] = *(r[Rb] + sext(d))
ST Ra,d(Rb)	01	*(r[Rb] + sext(d)) = r[Ra]



mnemonic	op2	function
(reserved)	000	
LI Rb,d	001	r[Rb] = sext(d)
(reserved)	010	
(reserved)	011	
BL d(Rb)	100	PC = r[Rb] + sext(d)
BAL Rb,d	101	r[Rb] = PC + 1 ; PC = PC + 1 + sext(d)
B d	110	PC = PC + 1 + sext(conc(Rb,d))

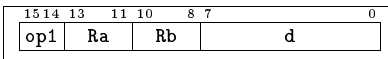


mnemonic	cond	function
BE d	000	if (Z) PC = PC + 1 + sext(d)
BLT d	001	if (S) PC = PC + 1 + sext(d)
BLE d	010	if (Z || S) PC = PC + 1 + sext(d)
BC d	011	if (C) PC = PC + 1 + sext(d)
BNE d	100	if (!Z) PC = PC + 1 + sext(d)
BGE d	101	if (!S) PC = PC + 1 + sext(d)
BGT d	110	if (!(Z || S)) PC = PC + 1 + sext(d)
BNC d	111	if (!C) PC = PC + 1 + sext(d)

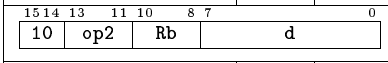


mnemonic	op3	function
ADD Rd,Rs	0000	r[Rd] = r[Rd] + r[Rs]
SUB Rd,Rs	0001	r[Rd] = r[Rd] - r[Rs]
AND Rd,Rs	0010	r[Rd] = r[Rd] & r[Rs]
OR Rd,Rs	0011	r[Rd] = r[Rd] | r[Rs]
XOR Rd,Rs	0100	r[Rd] = r[Rd] ^ r[Rs]
CMP Rd,Rs	0101	r[Rd] - r[Rs]
MOV Rd,Rs	0110	r[Rd] = r[Rs]
(reserved)	0111	
SLL Rd,d	1000	r[Rd] = shift_left_logical(r[Rd],d)
SLR Rd,d	1001	r[Rd] = shift_left_rotate(r[Rd],d)
SRL Rd,d	1010	r[Rd] = shift_right_logical(r[Rd],d)
SRA Rd,d	1011	r[Rd] = shift_right_arithmetic(r[Rd],d)
IN Rd,d	1100	r[Rd] = (d & 1) ? io_status : input)
OUT Rs	1101	output = r[Rs]
(reserved)	1110	
HLT	1111	halt()

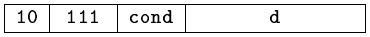
付録 B　拡張命令セット



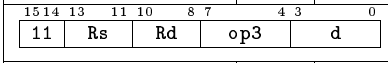
mnemonic		op1	function
LD	Ra,d(Rb)	00	r[Ra] = *(r[Rb] + sext(d))
ST	Ra,d(Rb)	01	*(r[Rb] + sext(d)) = r[Ra]



mnemonic		op2	function
TS	d(Rb)	000	Z = (*(r[Rb]+sext(d))==0) ; *(r[Rb] + sext(d)) = 1
LI	Rb,d	001	r[Rb] = sext(d)
LIL	Rb,d	010	r[Rb] = (r[Rb] & 0xFF00) | d
LIH	Rb,d	011	r[Rb] = (r[Rb] & 0x00FF) | (d << 8)
BL	d(Rb)	100	PC = r[Rb] + sext(d)
BAL	Rb,d	101	r[Rb] = PC + 1 ; PC = PC + 1 + sext(d)
B	d	110	PC = PC + 1 + sext(conc(Rb,d))



mnemonic		cond	function
BE	d	000	if (Z) PC = PC + 1 + sext(d)
BLT	d	001	if (S) PC = PC + 1 + sext(d)
BLE	d	010	if (Z || S) PC = PC + 1 + sext(d)
BC	d	011	if (C) PC = PC + 1 + sext(d)
BNE	d	100	if (!Z) PC = PC + 1 + sext(d)
BGE	d	101	if (!S) PC = PC + 1 + sext(d)
BGT	d	110	if (!(Z || S)) PC = PC + 1 + sext(d)
BNC	d	111	if (!C) PC = PC + 1 + sext(d)



mnemonic		op3	function
ADD	Rd,Rs/immd	0000	r[Rd] = r[Rd] + opd(i,Rs,d)
SUB	Rd,Rs/immd	0001	r[Rd] = r[Rd] - opd(i,Rs,d)
AND	Rd,Rs/immd	0010	r[Rd] = r[Rd] & opd(i,Rs,d)
OR	Rd,Rs/immd	0011	r[Rd] = r[Rd] | opd(i,Rs,d)
XOR	Rd,Rs/immd	0100	r[Rd] = r[Rd] ^ opd(i,Rs,d)
CMP	Rd,Rs/immd	0101	r[Rd] - opd(i,Rs,d)
MOV	Rd,Rs/immd	0110	r[Rd] = opd(i,Rs,d)
TRAP	Rs/immd	0111	raise(opd(i,Rs,d))
SLL	Rd,Rs/immd	1000	r[Rd] = shift_left_logical(r[Rd],opd(i,Rs,d))
SLR	Rd,Rs/immd	1001	r[Rd] = shift_left_rotate(r[Rd],opd(i,Rs,d))
SRL	Rd,Rs/immd	1010	r[Rd] = shift_right_logical(r[Rd],opd(i,Rs,d))
SRA	Rd,Rs/immd	1011	r[Rd] = shift_right_arithmetic(r[Rd],opd(i,Rs,d))
IN	Rd,Rs/immd	1100	r[Rd] = input[opd(i,Rs,d)]
OUT	Rd,Rs/immd	1101	output[opd(i,Rs,d)] = r[Rd]
LDSYS	Rd,sysreg	1110	r[Rd] = sysreg[conc(Rs,i,d)]
STSYS	Rd,sysreg	1111	sysreg[conc(Rs,i,d)] = r[Rd]