本テキストや授業のビデオなどの電子ファイルを他人に転送したり、ネットへアップロードすることなどを禁止します。

論理設計
東野担当12回目
授業スライド
１２月23日４限

基礎工学部情報科学科　東野輝夫

授業計画

- 授業計画：
  1. ドントケアを含む論理関数の簡単化（６章）
  2. フリップフロップとレジスタ（１０章）
  3. 同期式順序回路（Mealy型, Moore型順序回路）（１１章）
  4. カルノー図を用いた論理関数の簡単化（１章から５章の復習）
  5. 組合せ論理回路設計、よく用いられる組み合わせ回路（７章，８章）
  6. 加減算器とALU、順序回路の簡単化（９章，１２章前半）
  7. 演習
  8. 順序回路の簡単化、カウンタ（１２章後半，１３章）
  9. 中間試験（１章～１１章）
  10. ＩＣを用いた順序回路の実現（１５章）
  11. 演習
  12. ＣＰＵの設計（付録）
  13. ＣＰＵの設計，演習
  14. 乗算器と除算器（１４章）
  15. 期末試験（１２章～１５章，付録）

質問について

- メールで随時問い合わせや質問にお答えしますので，何かあれば，higashino@ist.osaka-u.ac.jp までメールで質問して下さい.
- また、時間を決めてZoomなどを用いて質問にお答えすることも可能ですので、まずはメールで疑問点や問い合わせ事項などを連絡して下さい。

お願い

本テキストや授業のビデオなどの電子ファイルを他人に転送したり、ネットへアップロードすることなどを禁止します。

著作権保護

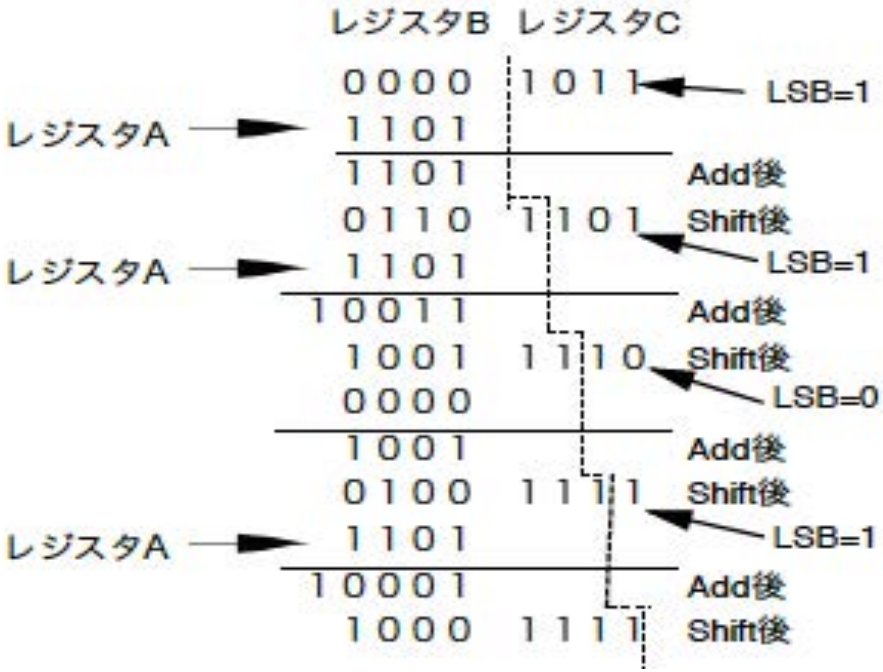
- この授業のテキスト（教科書）や授業スライド、授業ビデオの著作権保護に努めて下さい.
- この授業のビデオやスナップショットを録画したり，それらを他の人に転送したり，インターネット上で公開したりすることを禁止します.
- この授業で利用するスライドにはオーム社の教科書の図などが含まれているので，著作権保護の観点から，この授業スライドの公開につながる行為は謹んでください.
- 来年度は CLE を使ったメディア授業でなく，対面の授業ができることを期待していますが，今年度の演習課題の解答が事前に公開されたりすると，来年度の授業で同じ演習課題が使えなくなり，授業テキストの大幅な修正が必要になるため，協力をお願いします.

レポート課題の解答（演習問題２）

演習問題２

与えられた数n の平方根√n を求める同期式順序回路を設計せよ

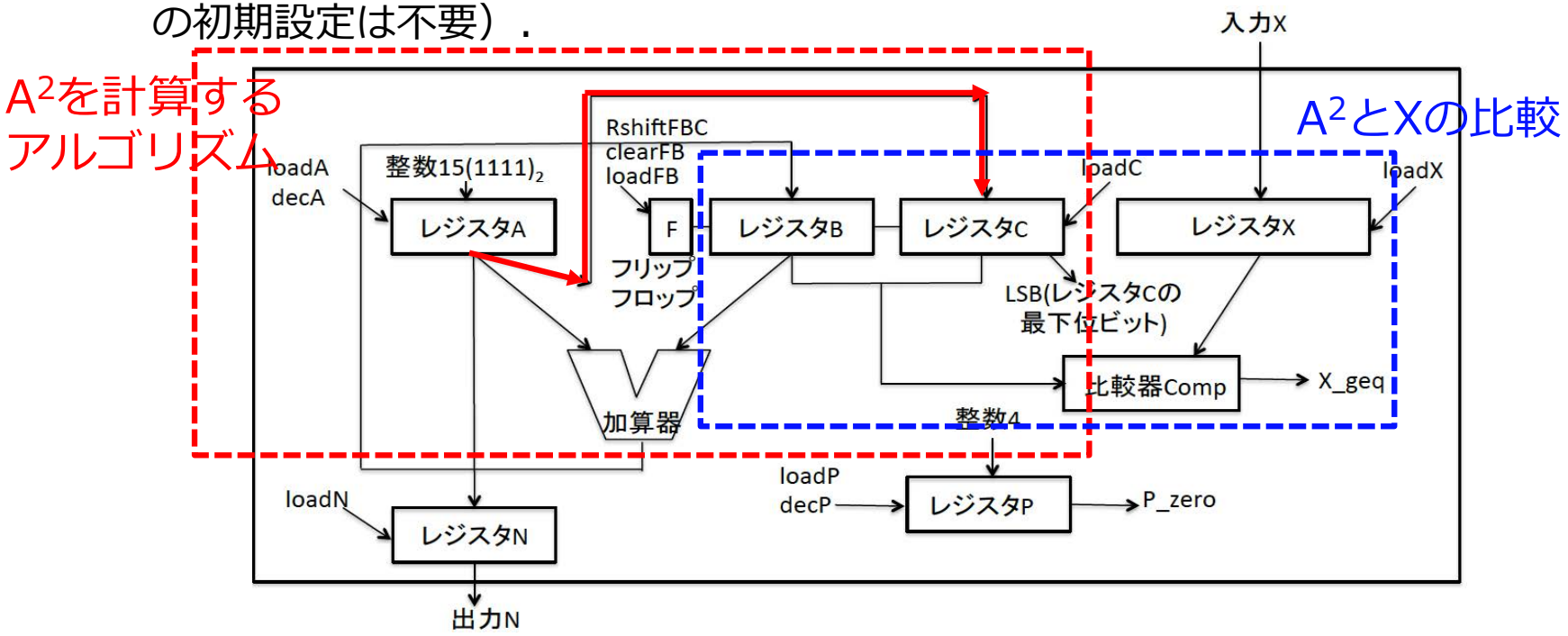
- レジスタAとレジスタCの積の計算は，下図のようにAdd（加算），Shift（右に１ビットシフト）を繰り返して行う（LSBが １ のときレジスタAの値をレジスタBに加算する）．最終的にレジスタAとレジスタCの積がレジスタB，レジスタCに格納される（レジスタBが上位4ビット，レジスタCが下位4ビットになる）．これは演習問題１で説明したかけ算のアルゴリズムと同じです．



演習問題２

与えられた数n の平方根√n を求める同期式順序回路を設計せよ

- 下図のような回路を用いて，外部から入力された 8 ビットの非負整数Xに対して，X の平方根の近似値（$(N+1)^2 > X \geqq N^2$ となる 4 ビットの非負整数 N）を求める同期式順序回路を作成したい．図のレジスタA とレジスタC に最初 4 ビットの最大値 $15(1111)_2$ を N の値としてセットし，レジスタP に整数値 4 をセットする．レジスタA とレジスタC の積 N^2 を計算し，その値とレジスタX の値を比較する．レジスタX の値が N^2 以上（$X \geqq N^2$）になれば比較器 Comp の X_geq信号 が 1 になり，その時のレジスタA の値を X の平方根の近似値としてレジスタN にセットする（レジスタN の初期設定は不要）．



演習問題２

- 各部品は次のように動作するものと仮定する.

(a) loadA=1, decA=0のとき 4ビットの最大値15($1111)_2$をレジスタAに格納（A←15）
loadA=0, decA=1のとき レジスタAの値を１減らす（A←A-1）
loadA=0, decA=0のとき レジスタAの値は不変

(b) RshiftFBC=1, clearFB=0, loadFB=0, loadC=0, decC=0 のとき
F,B,Cの値を1ビット右にシフトする（Rshift(F,B,C) と書く）
RshiftFBC=0, clearFB=1, loadFB=0 のときF,Bの値を0にする（<F,B>←0）
RshiftFBC=0, clearFB=0, loadFB=1 のとき(A+B)を<F,B>に格納（<F,B>←A+B）
RshiftFBC=0, clearFB=0, loadFB=0, loadC=0, decC=0 のときF,B,Cの値は不変

(c) loadC=1のとき レジスタAの値をレジスタCに格納する（C←A）
loadC=0のとき レジスタCの値は不変

(d) loadX=1のとき 8ビットの入力Xをレジスタに格納する（X←input(X)）
loadX=0のとき レジスタXの値は不変

(e) loadP=1, decP=0 のとき 整数4をレジスタPに格納する（P←4）
loadP=0, decP=1 のとき レジスタPの値を１減らす（P←P-1）
loadP=0, decP=0 のとき レジスタPの値は不変

(f) loadN=1のとき レジスタAの値をレジスタNに格納する（N←A）
loadN=0のとき レジスタNの値は不変

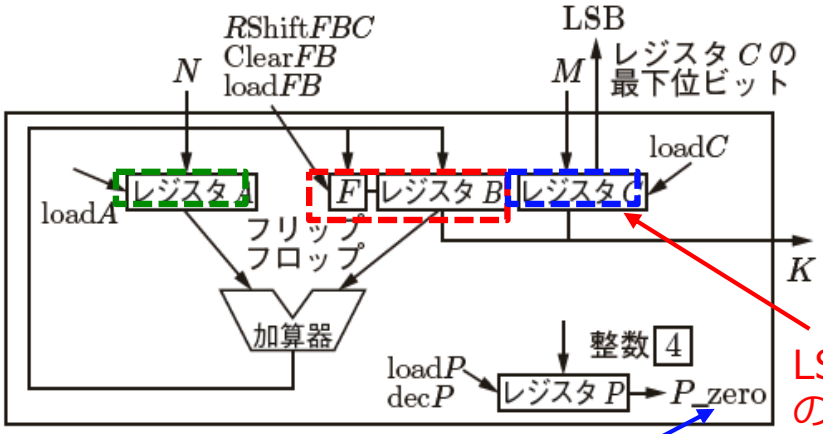
(g) P=0 が真のときかつそのときのみ P_zero=1 （test(P=0)? と書く）

(h) X≧A*C(=<B,C>) が真のときかつそのときのみ X_geq=1 （test(X≧A*C)? と書く）

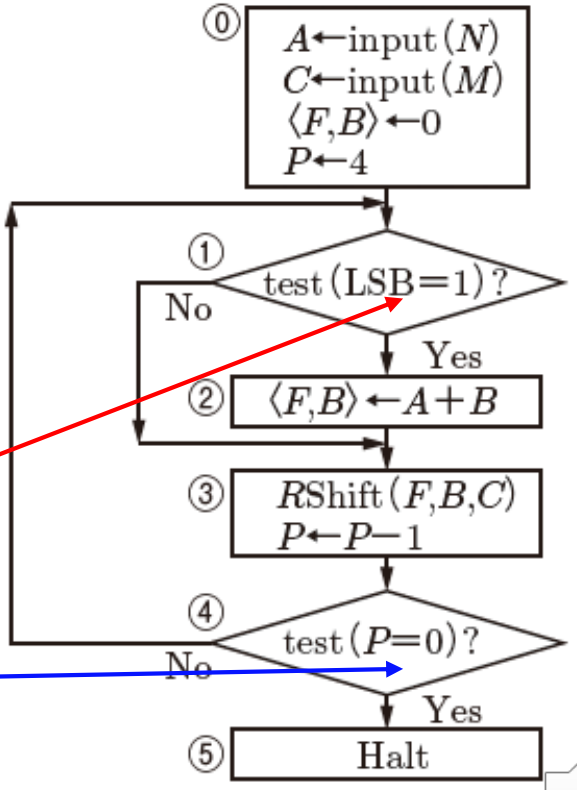
(i) レジスタCの最下位ビットが1のとき かつそのときのみ LSB=1（test(LSB=1)? と書く）

二つの整数の積を計算する同期式順序回路

- 電卓の回路や自動販売機の制御回路と同様の方法で，二つの整数の積を計算する同期式順序回路を設計せよ
  - ２進数の掛け算（1101＊1011＝10001111＝13*11=143）



解図 15・1　非負整数 N, M の積 K を求める同期式順序回路



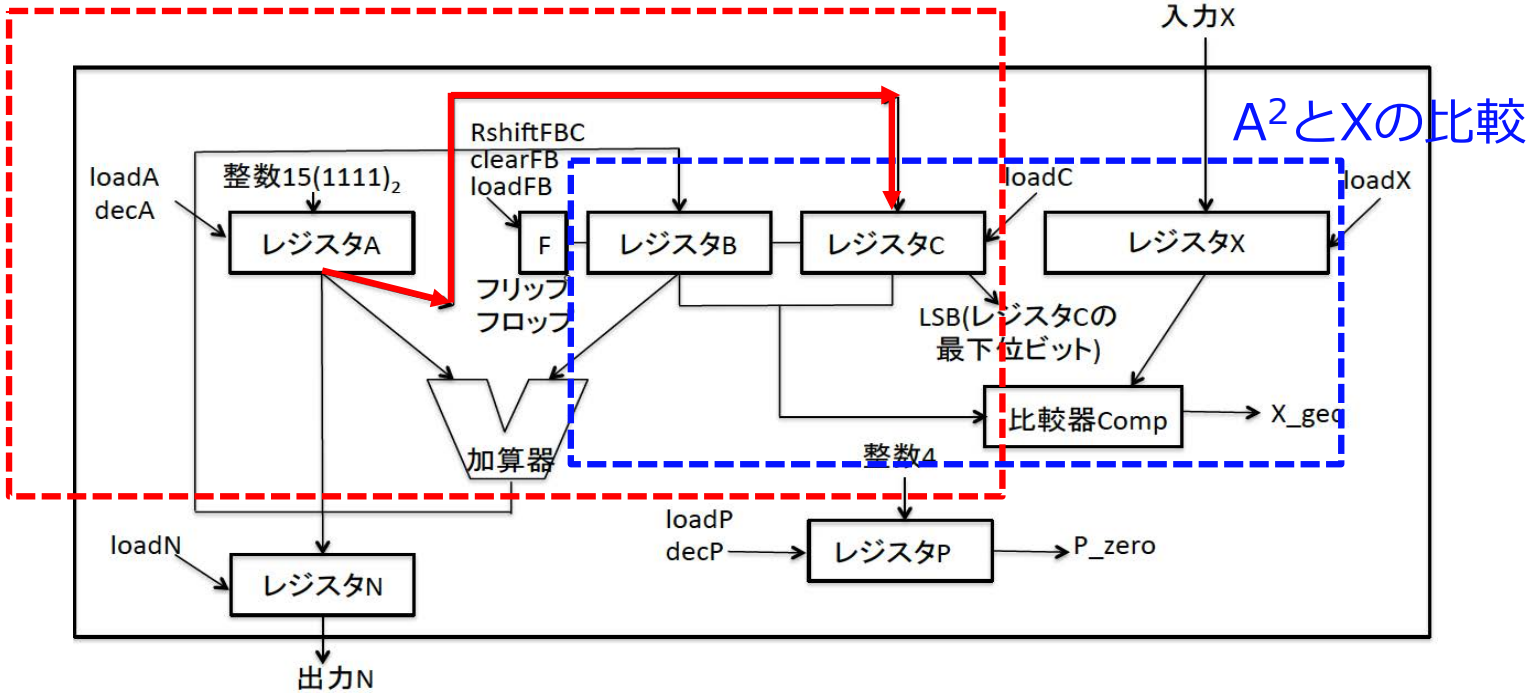
解図 15・4　非負整数 N, M の積 K を求めるアルゴリズムのフローチャート

演習問題２

（問題１）

– (a)～(i)の動作仕様をもとに，非負整数Xの平方根の近似値（$(N+1)^2 > X \geqq N^2$となるN）を求めるアルゴリズムのフローチャートを考え，下図のフローチャートの空白を埋めよ.

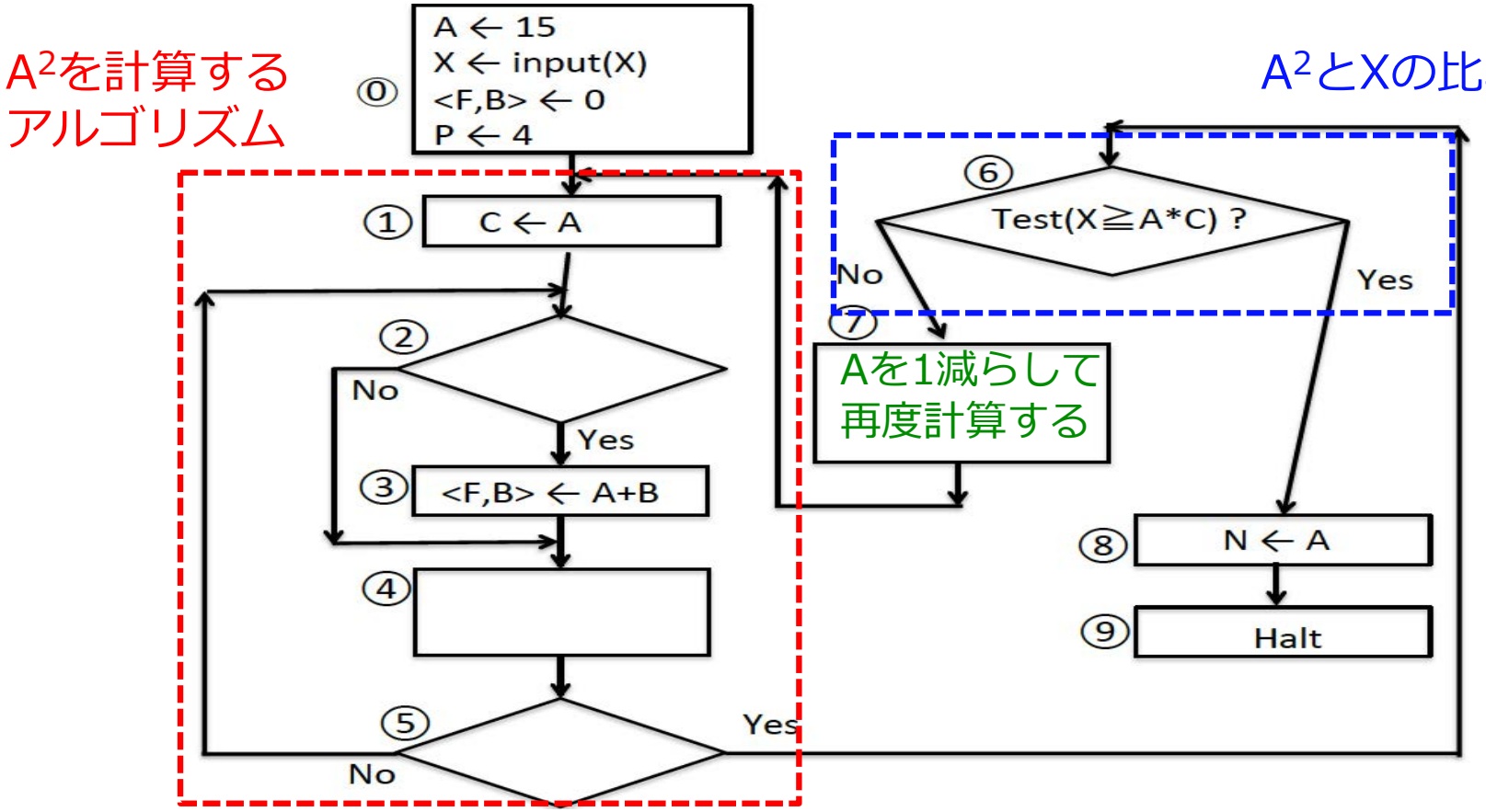
A^2を計算する
アルゴリズム



演習問題２

（問題１）

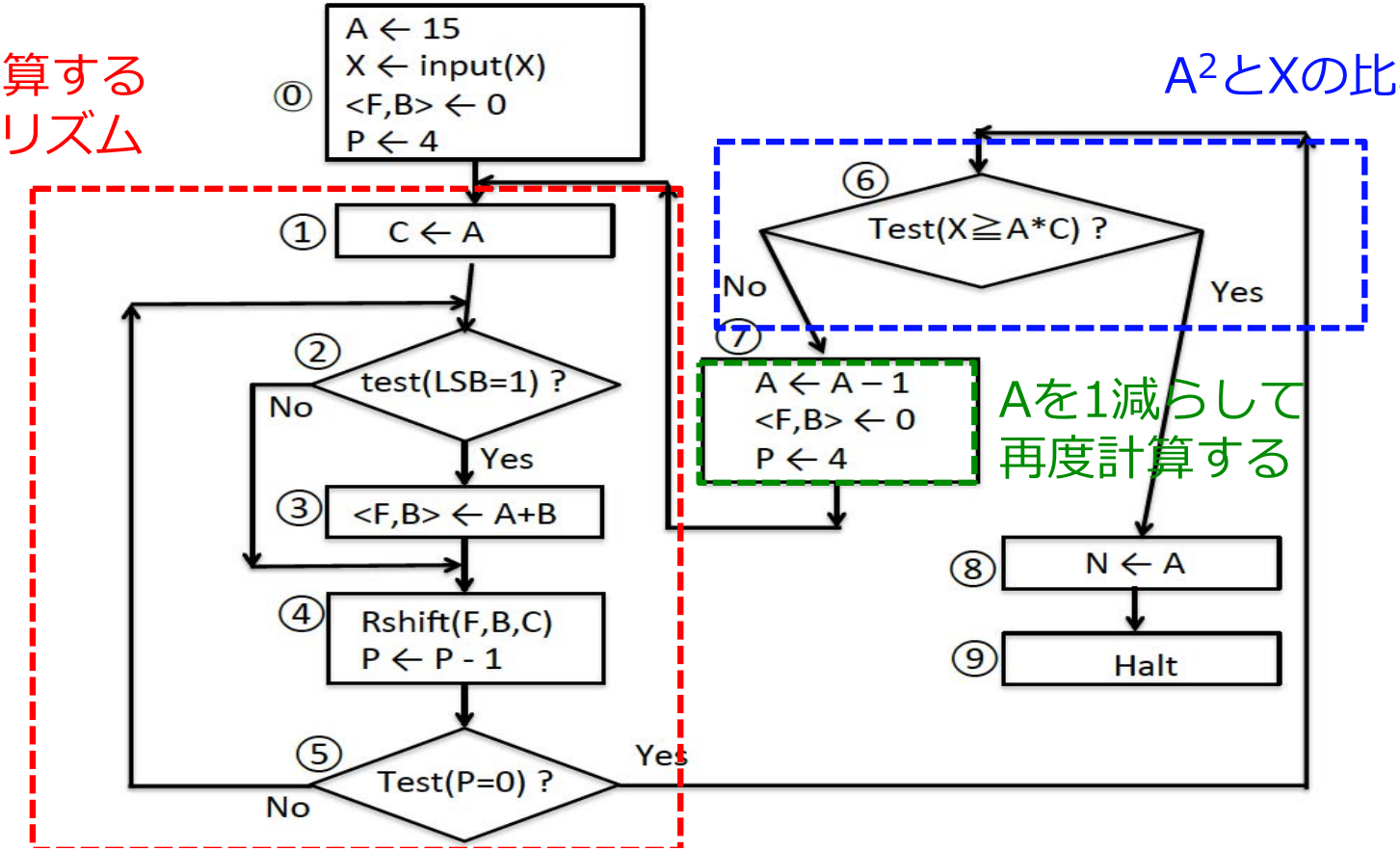
– (a)～(i)の動作仕様をもとに，非負整数Xの平方根の近似値（$(N+1)^2 > X \geqq N^2$となるN）を求めるアルゴリズムのフローチャートを考え，下図のフローチャートの空白を埋めよ.



演習問題２解答

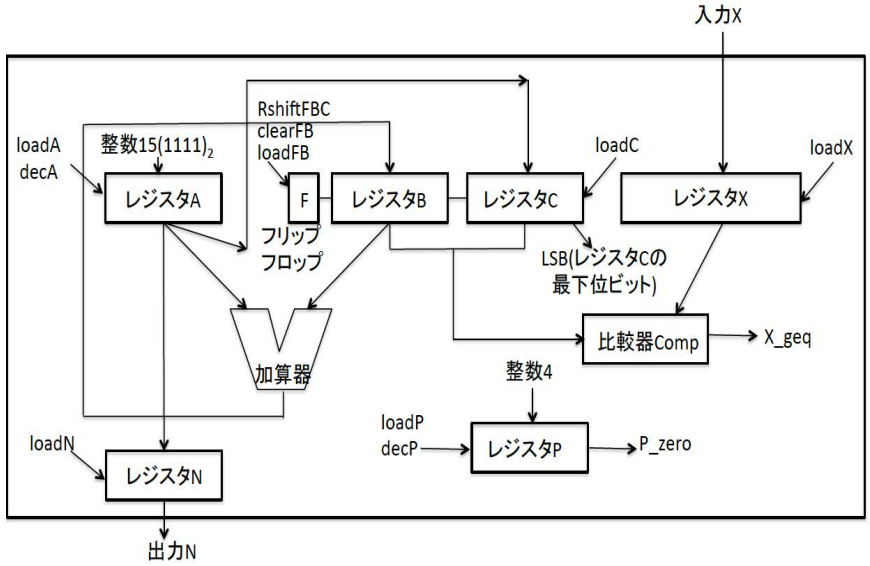
（問題１）

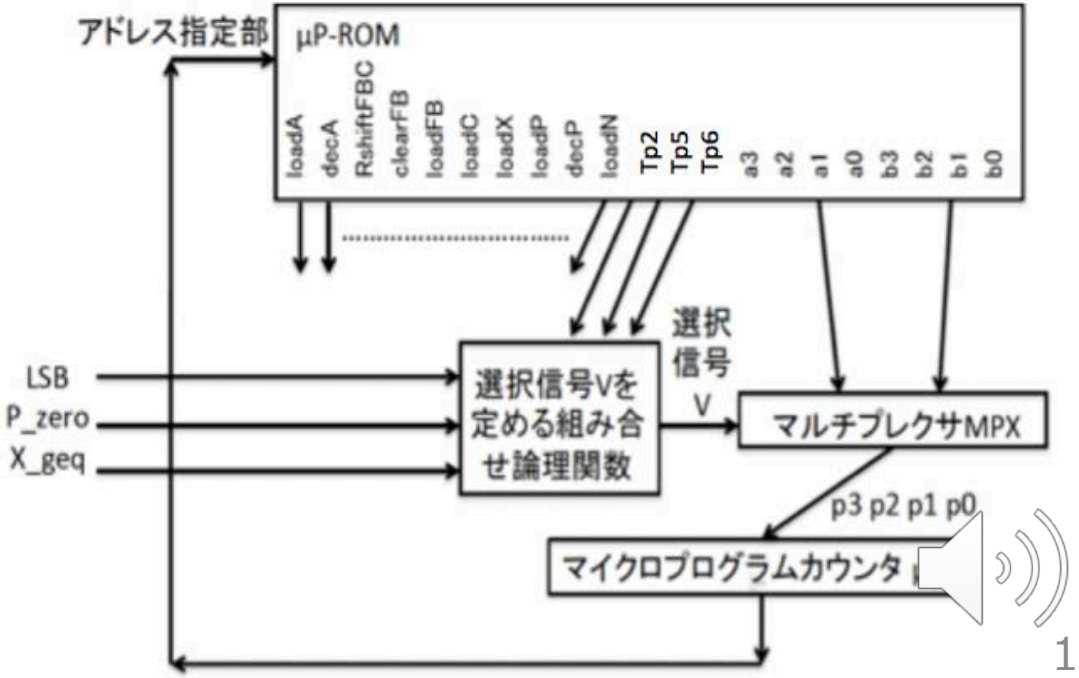
– (a)～(i)の動作仕様をもとに，非負整数Xの平方根の近似値（$(N+1)^2 > X \geqq N^2$となるN）を求めるアルゴリズムのフローチャートを考え，下図のフローチャートの空白を埋めよ.



演習問題２

- 左下図の順序回路の制御部を右下図のような制御回路で実現したい．右下図のマイクロプログラムカウンタμPCの値 $p_3p_2p_1p_0$（p_0が最下位ビット）はμP-ROM のアドレスを表す．
- マルチプレクサ MPX は選択信号 V の値が 0 のとき $a_3a_2a_1a_0$ を出力し，V の値が 1 のとき $b_3b_2b_1b_0$ を出力する．また，制御信号 loadA, decA, RshiftFBC, clearFB, loadFB, loadC, loadX, loadP, decP, loadN の値は，マイクロプログラムカウンタ μPC の値 p_3,p_2,p_1,p_0 をアドレスとしてμP-ROM にその信号値を保存することで実現する．



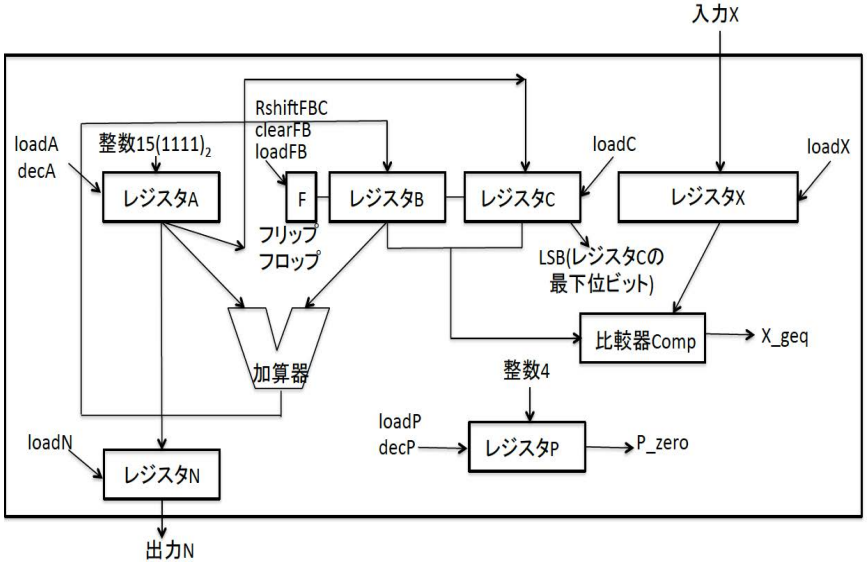


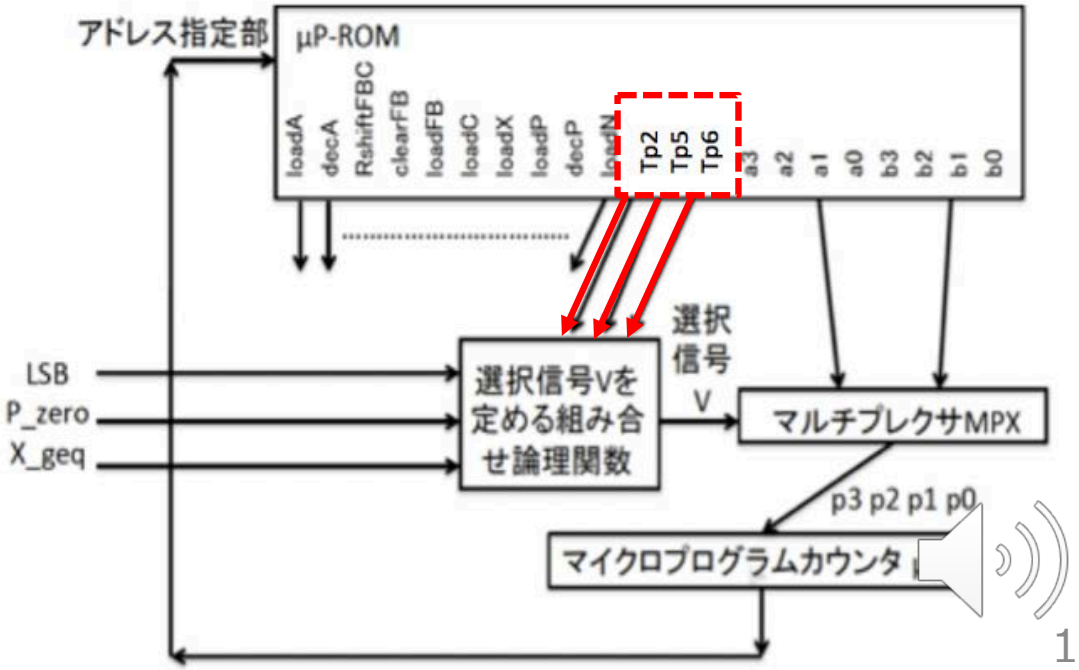
演習問題２

(問題2-1) 制御信号 loadA, decA, RshiftFBC の値をそれぞれ p_3,p_2,p_1,p_0 を引数とする論理式で表せ.

(問題2-2) 選択信号 V の値を T_{p2},T_{p5},T_{p6}, LSB, P_zero, X_geq の論理式で表せ（条件分岐 ②, ⑤, ⑥ に対応して T_{p2},T_{p5},T_{p6} を用いよ）.

(問題2-3) この回路のμP-ROM の内容とその 0 番地から Z 番地 (Z≦9) の内容を定めよ. ドント・ケアでよい部分は×印を付けよ.





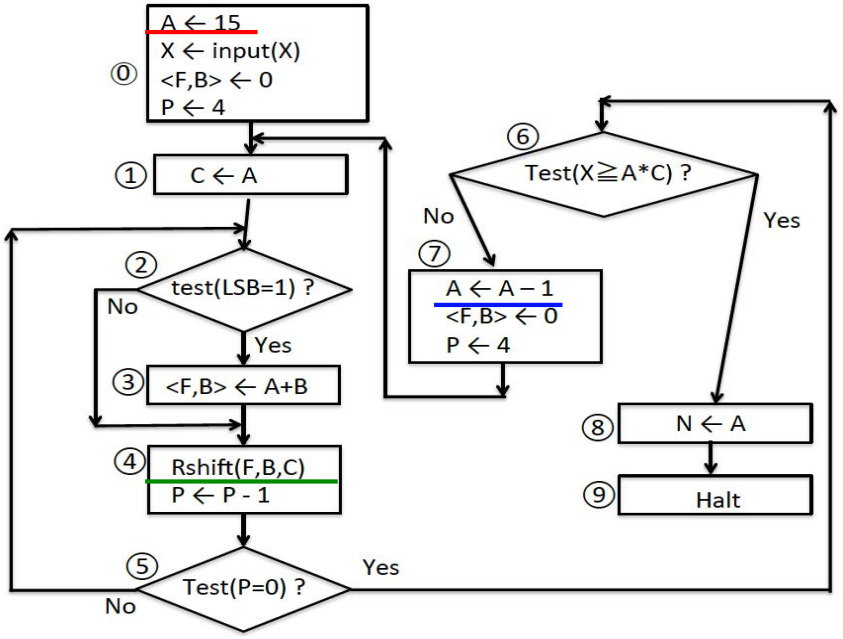
演習問題２解答

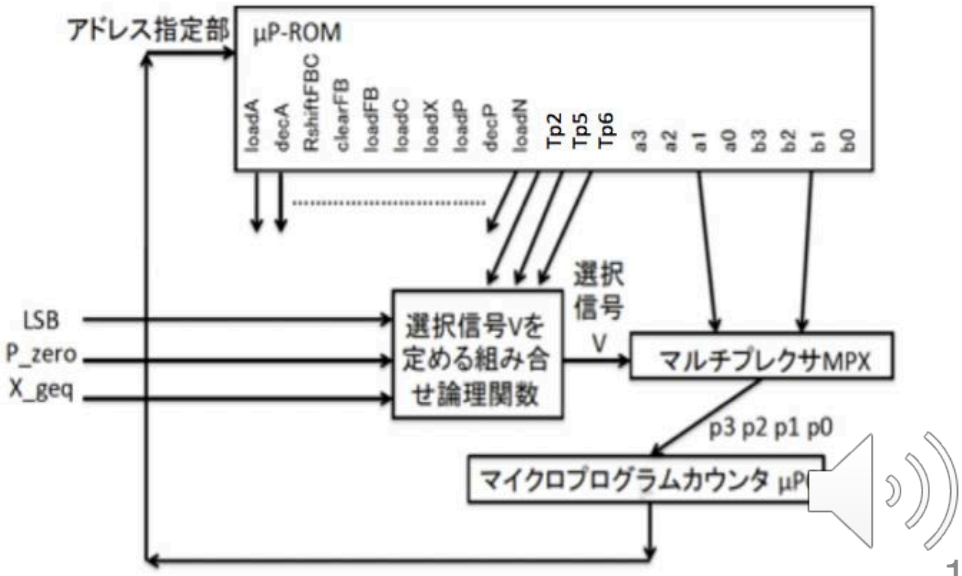
(問題2-1) 制御信号 loadA, decA, RshiftFBC の値をそれぞれ p_3,p_2,p_1,p_0 を引数とする論理式で表せ.

（解答）

(2-1) 制御信号

- loadA ＝ $\neg p_3 \cdot \neg p_2 \cdot \neg p_1 \cdot \neg p_0$ （$<p_3,p_2,p_1,p_0> = <0,0,0,0>$のとき）
- decA ＝ $\neg p_3 \cdot p_2 \cdot p_1 \cdot p_0$ （$<p_3,p_2,p_1,p_0> = <0,1,1,1>$のとき）
- RshiftFBC＝ $\neg p_3 \cdot p_2 \cdot \neg p_1 \cdot \neg p_0$ （$<p_3,p_2,p_1,p_0> = <0,1,0,0>$のとき）





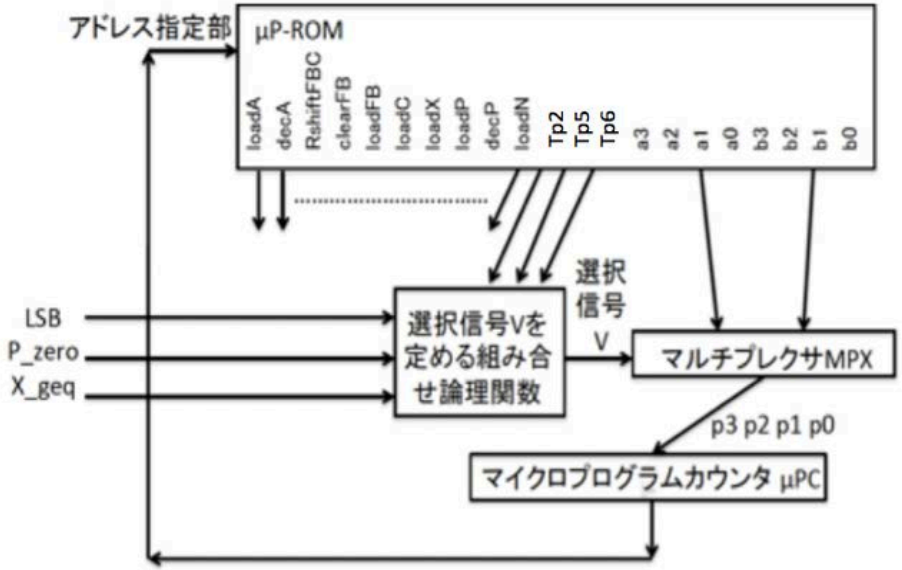
演習問題２解答

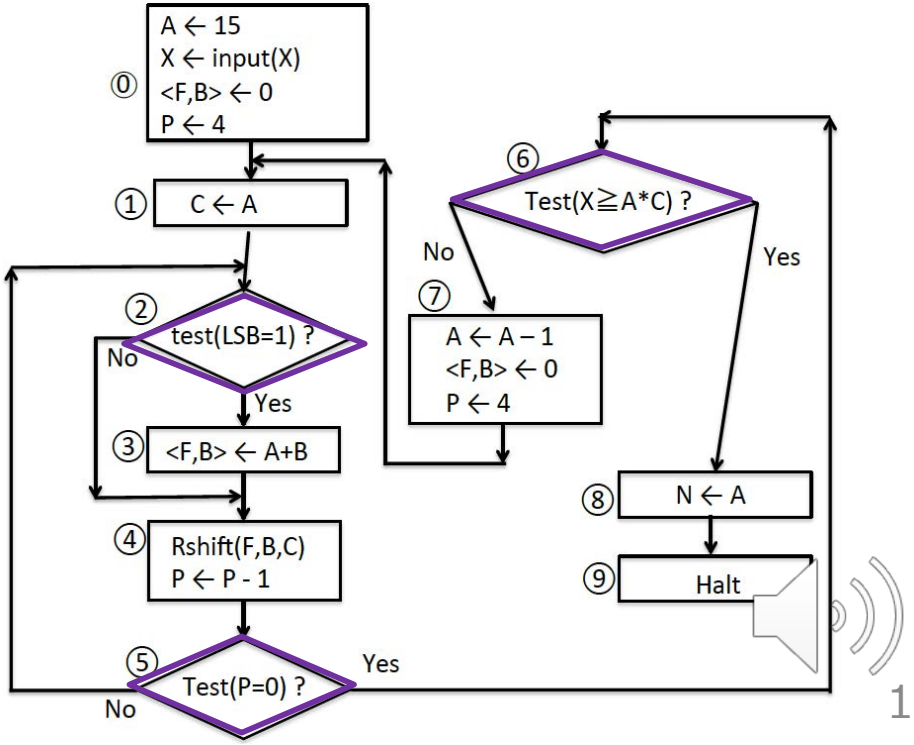
(問題2-2) 選択信号 V の値を T_{p2},T_{p5},T_{p6}, LSB, P_zero, X_geq の論理式で表せ（条件分岐 ②, ⑤, ⑥ に対応して T_{p2},T_{p5},T_{p6} を用いよ）.

（解答）

(2-2) 選択信号

– $V = (Tp_2 \cdot LSB) \vee (Tp_5 \cdot P\_zero) \vee (Tp_6 \cdot X\_geq)$





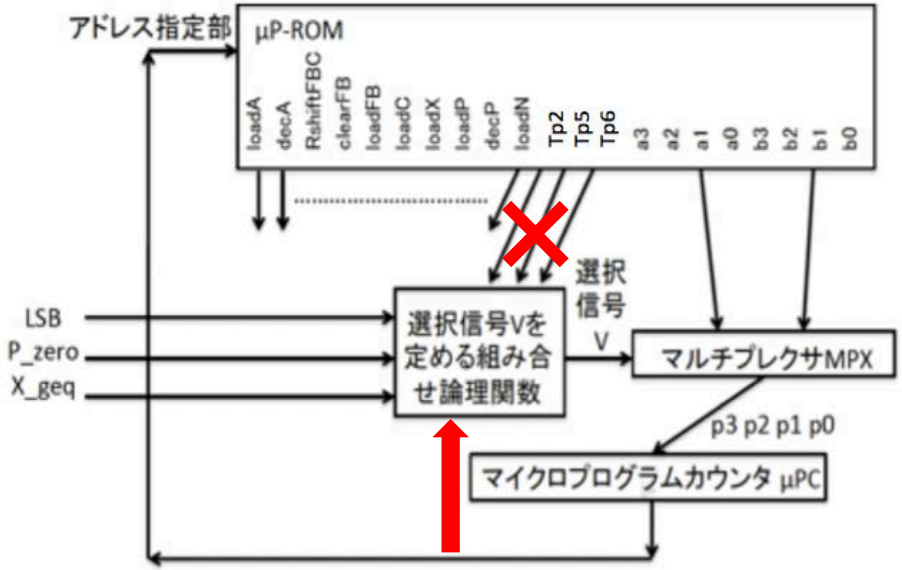
演習問題２解答

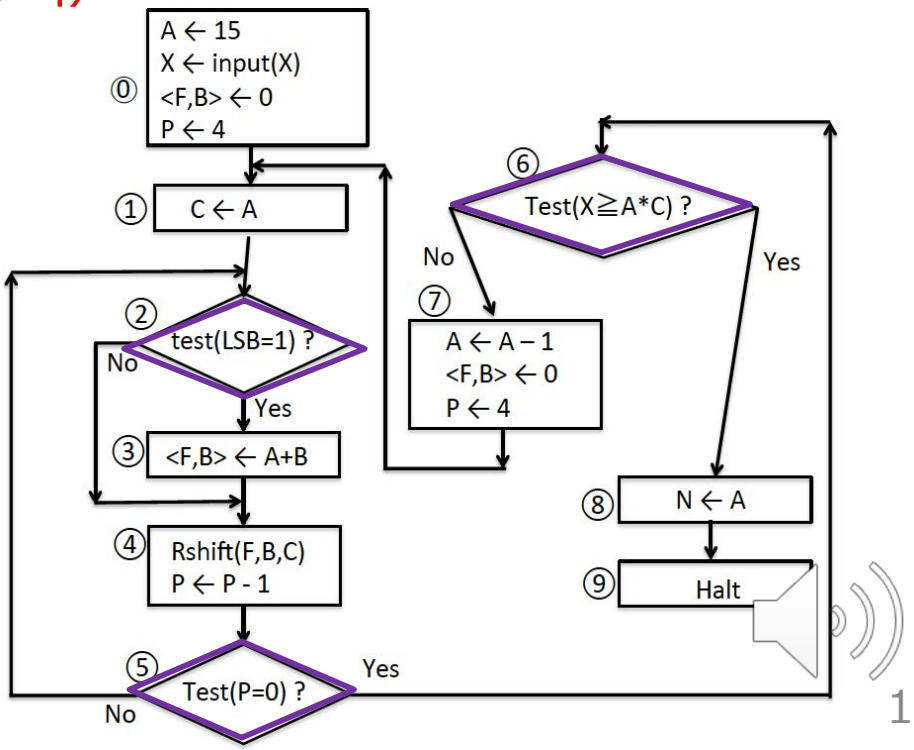
(問題2-2) 選択信号 V の値を T_{p2},T_{p5},T_{p6}, LSB, P_zero, X_geq の論理式で表せ（条件分岐 ②, ⑤, ⑥ に対応して T_{p2},T_{p5},T_{p6} を用いよ）.

(2-2) 選択信号

– $V = (Tp_2 \cdot LSB) \vee (Tp_5 \cdot P\_zero) \vee (Tp_6 \cdot X\_geq)$

$= ((\neg p_3 \cdot \neg p_2 \cdot p_1 \cdot \neg p_0) \cdot LSB) \vee ((\neg p_3 \cdot p_2 \cdot \neg p_1 \cdot p_0) \cdot P\_zero)$
$\vee ((\neg p_3 \cdot p_2 \cdot p_1 \cdot \neg p_0) \cdot X\_geq)$



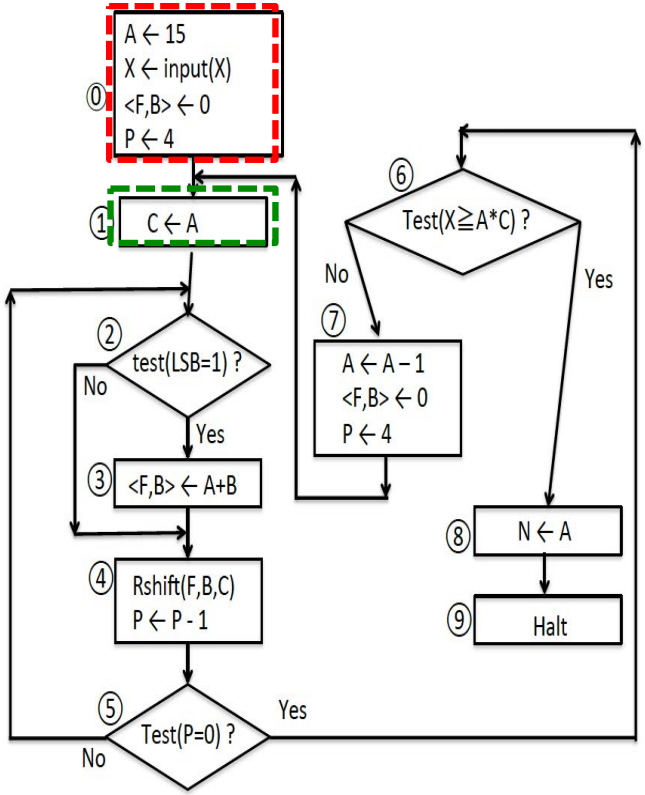


演習問題２解答

(問題2-3)

この回路のμP-ROM の内容とその 0 番地から Z 番地 (Z≦9) の内容を定めよ．ドント・ケアでよい部分は×印を付けよ．

	p3 p2 p1 p0	loadA	decA	RshiftFBC	clearFB	loadFB	loadC	loadX	loadP	decP	loadN	Tp2	Tp5	Tp6	a3	a2	a1	a0	b3	b2	b1	b0
0番地	0 0 0 0	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	1	×	×	×	×
1番地	0 0 0 1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	×	×	×	×
2番地	0 0 1 0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1
3番地	0 0 1 1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	×	×	×	×
4番地	0 1 0 0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	×	×	×	×
5番地	0 1 0 1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1	0
6番地	0 1 1 0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	0	0
7番地	0 1 1 1	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	×	×	×	×
8番地	1 0 0 0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	×	×	×	×
9番地	1 0 0 1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	×	×	×	×

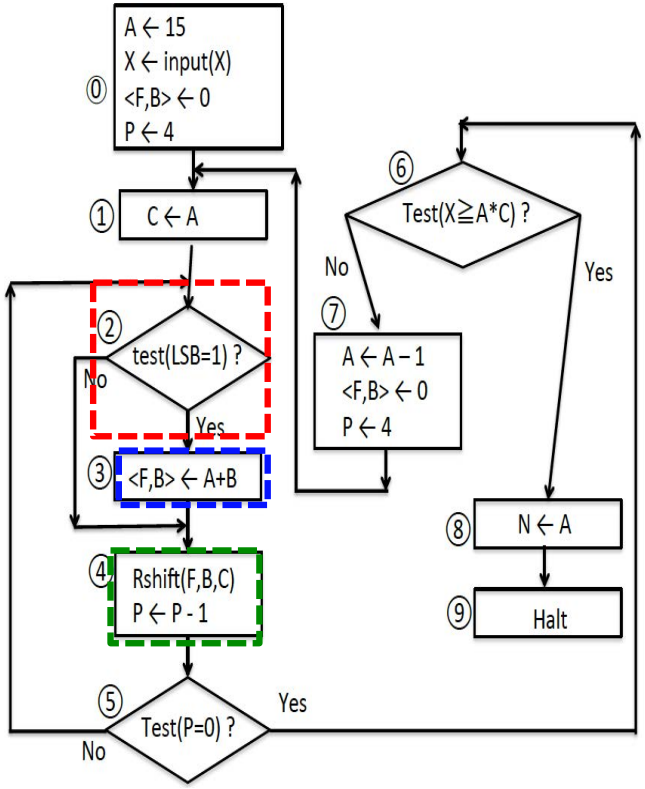


演習問題２解答

(問題2-3)

この回路のμP-ROM の内容とその 0 番地から Z 番地 (Z≦9) の内容を定めよ．ドント・ケアでよい部分は×印を付けよ．

	p3 p2 p1 p0	loadA	decA	RshiftFBC	clearFB	loadFB	loadC	loadX	loadP	decP	loadN	Tp2	Tp5	Tp6	a3	a2	a1	a0	b3	b2	b1	b0
0番地	0 0 0 0	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	1	×	×	×	×
1番地	0 0 0 1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	×	×	×	×
2番地	0 0 1 0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1
3番地	0 0 1 1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	×	×	×	×
4番地	0 1 0 0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	×	×	×	×
5番地	0 1 0 1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1	0
6番地	0 1 1 0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	0	0
7番地	0 1 1 1	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	×	×	×	×
8番地	1 0 0 0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	×	×	×	×
9番地	1 0 0 1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	×	×	×	×

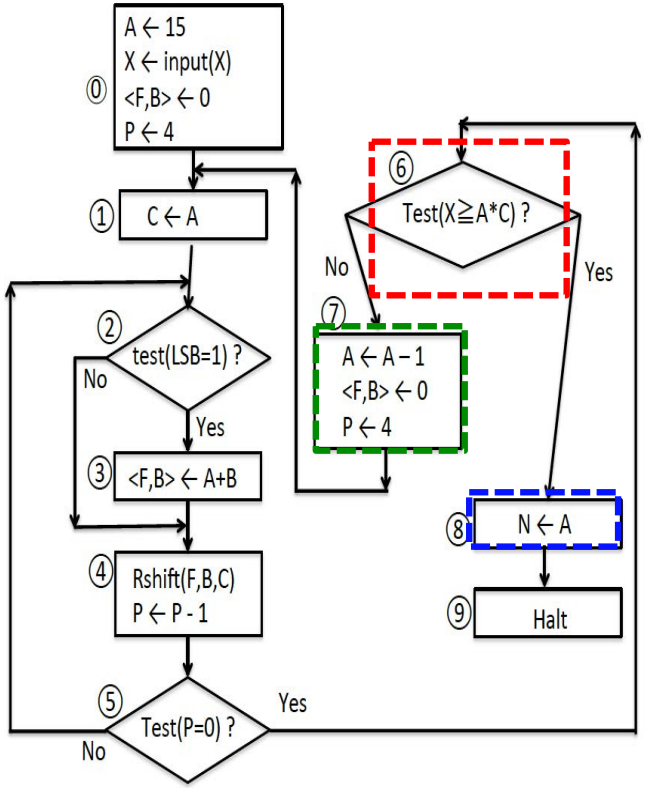


演習問題２解答

(問題2-3)

この回路のμP-ROM の内容とその 0 番地から Z 番地 (Z≦9) の内容を定めよ．ドント・ケアでよい部分は×印を付けよ．

	p3 p2 p1 p0	loadA	decA	RshiftFBC	clearFB	loadFB	loadC	loadX	loadP	decP	loadN	Tp2	Tp5	Tp6	a3	a2	a1	a0	b3	b2	b1	b0
0番地	0 0 0 0	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	1	×	×	×	×
1番地	0 0 0 1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	×	×	×	×
2番地	0 0 1 0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1
3番地	0 0 1 1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	×	×	×	×
4番地	0 1 0 0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	×	×	×	×
5番地	0 1 0 1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1	0
6番地	0 1 1 0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	0	0
7番地	0 1 1 1	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	×	×	×	×
8番地	1 0 0 0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	×	×	×	×
9番地	1 0 0 1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	×	×	×	×

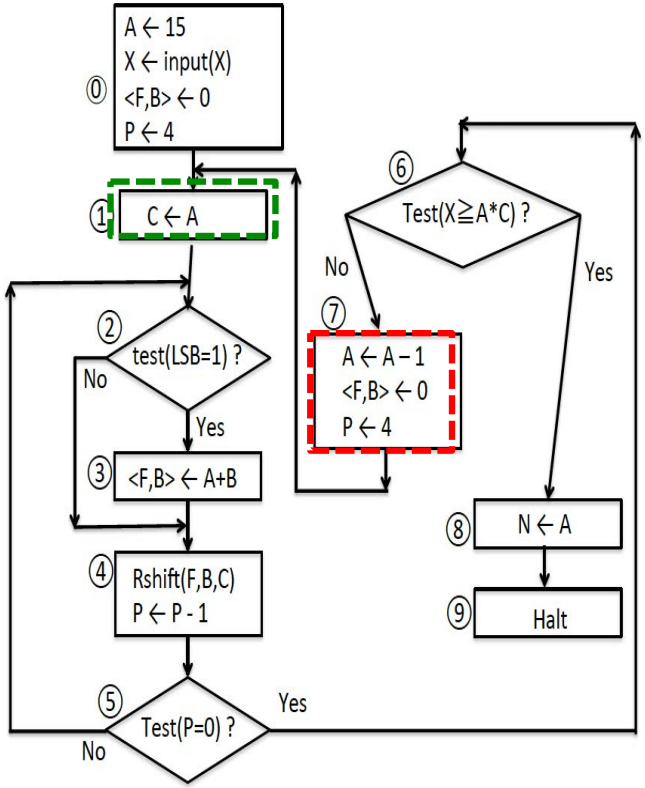


演習問題２解答

(問題2-3)

この回路のμP-ROM の内容とその 0 番地から Z 番地 (Z≦9) の内容を定めよ．ドント・ケアでよい部分は×印を付けよ．

	p3 p2 p1 p0	loadA	decA	RshiftFBC	clearFB	loadFB	loadC	loadX	loadP	decP	loadN	Tp2	Tp5	Tp6	a3	a2	a1	a0	b3	b2	b1	b0
0番地	0 0 0 0	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	1	×	×	×	×
1番地	0 0 0 1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	×	×	×	×
2番地	0 0 1 0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1
3番地	0 0 1 1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	×	×	×	×
4番地	0 1 0 0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	×	×	×	×
5番地	0 1 0 1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1	0
6番地	0 1 1 0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	0	0
7番地	0 1 1 1	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	×	×	×	×
8番地	1 0 0 0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	×	×	×	×
9番地	1 0 0 1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	×	×	×	×

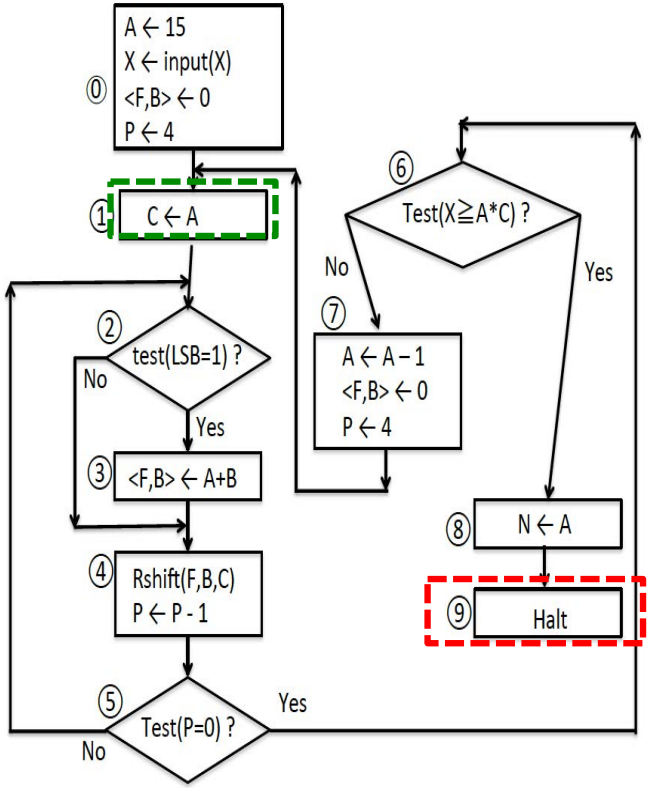


演習問題２解答

(問題2-3)

この回路のμP-ROM の内容とその 0 番地から Z 番地 (Z≦9) の内容を定めよ．ドント・ケアでよい部分は×印を付けよ．

	p3 p2 p1 p0	loadA	decA	RshiftFBC	clearFB	loadFB	loadC	loadX	loadP	decP	loadN	Tp2	Tp5	Tp6	a3	a2	a1	a0	b3	b2	b1	b0
0番地	0 0 0 0	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	1	×	×	×	×
1番地	0 0 0 1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	×	×	×	×
2番地	0 0 1 0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1
3番地	0 0 1 1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	×	×	×	×
4番地	0 1 0 0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	×	×	×	×
5番地	0 1 0 1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1	0
6番地	0 1 1 0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	0	0	0
7番地	0 1 1 1	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	1	×	×	×	×
8番地	1 0 0 0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	×	×	×	×
9番地	1 0 0 1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	×	×	×	×



付録
CPUの設計

付録
CPUの設計

- この章のねらい

付録　CPU の設計

ここでは，メモリやレジスタ，加算器，比較回路などの IC 部品を組み合わせて，簡単な CPU を同期式順序回路として実現するための方法について述べる．

設計するCPU の概要

- 機械語命令の指定
  - CPU（Central Processing Unit）はコンピュータの中心的な回路であり，中央処理装置とも呼ばれ，機械語で書かれたプログラムを実行することで，さまざまな数値計算や情報処理を行う．機械語の命令には，ロード（LD），ストア（ST），加算（AD），条件付ジャンプ（JZ）などいくつかの種類がある．
  - 本章では四つの命令（LD, ST, AD, JZ）からなるCPU の設計を考える．以下では，各命令は下記のような2 語命令（各inC_k = 2）とし，図A·1 のようにメモリの最初の 1 語にその命令の名前が書かれ，次の 1 語にその命令のオペランド（番地）が書かれているものとする．

	名前	オペランド	機能
1	LD	ADR	AC←M[ADR]，PC←PC+inC_1
2	ST	ADR	M[ADR]←AC，PC←PC+inC_2
3	AD	ADR	AC←AC+M[ADR]，PC←PC+inC_3
4	JZ	B	if AC=0 then PC←B else PC←PC+inC_4

（inC_k: 命令 k の占める語数，M[ADR]:メモリ M の ADR番地の内容）

設計するCPU の概要

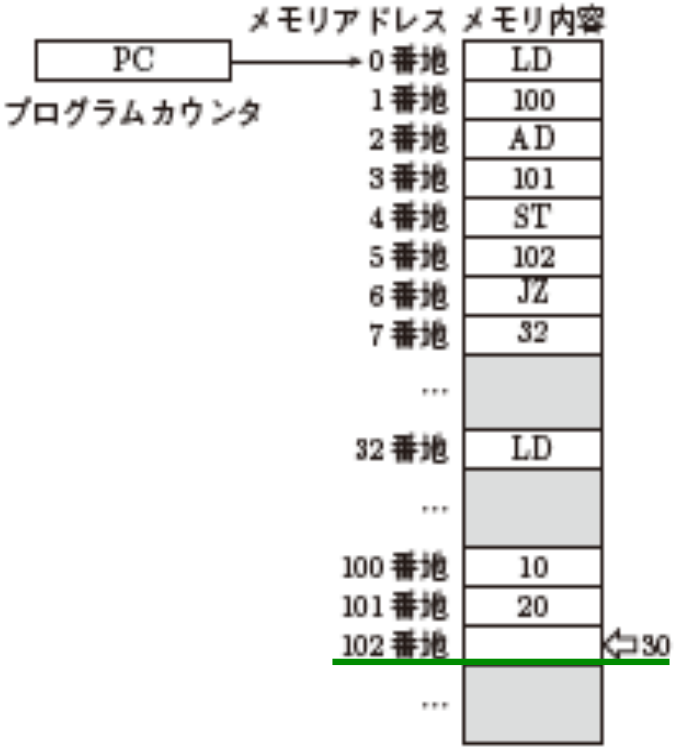
- 機械語命令の指定
  - ロード（LD）命令はメモリM のADR 番地の内容M[ADR] をアキュムレータ（レジスタ）AC に転送する命令
  - ストア（ST）命令は，アキュムレータAC の内容をメモリM のADR 番地に格納する命令
  - 加算（AD）命令は，アキュムレータAC の内容にメモリMのADR 番地の内容M[ADR]を加算する命令
  - これら三つの命令の実行後にプログラムカウンタPC の内容がそれぞれ 2 語（inC_k = 2, k = 1, 2, 3）増加する
  - 条件付ジャンプ（JZ）命令は，AC=0 が成り立つときのみプログラムカウンタPC の値を B にセットし，そうでなければプログラムカウンタの値を 2 語（inC_4 = 2）増加させる

	名前	オペランド	機能
1	LD	ADR	AC←M[ADR]，PC←PC+inC_1
2	ST	ADR	M[ADR]←AC，PC←PC+inC_2
3	AD	ADR	AC←AC+M[ADR]，PC←PC+inC_3
4	JZ	B	if AC=0 then PC←B else PC←PC+inC_4

（inC_k: 命令 k の占める語数，M[ADR]:メモリ M の ADR番地の内容）

メモリに格納された機械語プログラム（アセンブリ言語で表現）

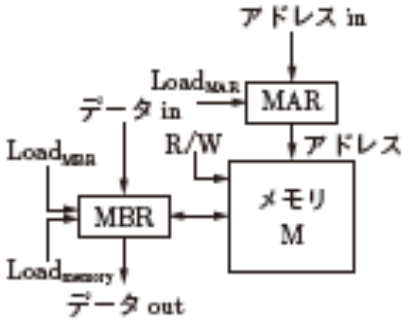
- 図A·1 にメモリに格納された機械語プログラムの例を示す．この例では，最初プログラムカウンタ PC は 0 番地を指しているものとする．0 番地の命令がロード（LD）命令で 1 番地に 100 が書かれているので，この 2 語命令を実行すると，100 番地の内容 10 がアキュムレータ AC に格納され，プログラムカウンタ PC の値が 2 増加する．
- PC が 2 番地の AD 命令を指し 3 番地に 101 が書かれているので，AD 命令が実行され，アキュムレータ AC に 101 番地の内容 20 が加算される（30 に変化）．
- その後プログラムカウンタ PC の値が 2 増加して 4 になり，5 番地に 102 が書かれているので，4 番地の ST 命令が実行され，アキュムレータ AC の値 30 が 102 番地に格納される．この命令が実行された後，プログラムカウンタ PC の値が 2 増加して 6 になる．
- 6 番地の命令は条件付ジャンプ（JZ）命令なので，AC = 0 かどうかがチェックされる．AC の値は 30 で条件（AC = 0）は偽になるので，7 番地に書かれた 32 番地にはジャンプせず，プログラムカウンタ PC の値はこれまでと同じように 2 だけ増加して 8 になり，8 番地以降の機械語が順次実行されていく．



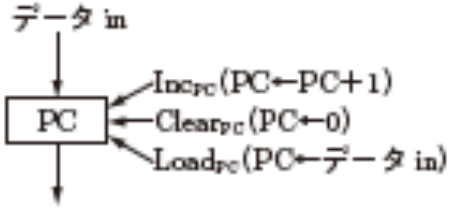
図A・1 メモリに格納された機械語プログラム（アセ…で表現）

利用するIC 部品

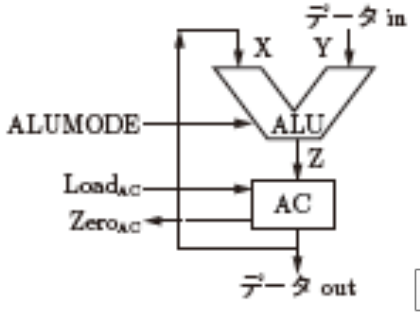
- 上記の四つの命令を実行する CPU を次のような機能をもつメモリ（図A·2），プログラムカウンタ（PC）（図A·3），算術論理演算装置（Arithmetic Logic Unit:ALU）（図A·4），命令レジスタ（Instruction Register: IR）などの部品を用いて実現する.
  - メモリM（図A·2）はR/W 信号が1 のとき，かつそのときのみ，メモリバッファレジスタMBR の内容がメモリM のMAR 番地M[MAR] に書き込まれる.
    （R/W=1 のとき　M[MAR]←MBR）
  - $Load_{memory}$ 信号を 1 にすると，メモリM の MAR 番地の内容 M[MAR] がメモリバッファレジスタ MBR に書き込まれる．また，$Load_{MBR}$, $Load_{MAR}$ 信号を 1 にすると，“データin” や“アドレスin” の内容がそ れぞれメモリバッファレジスタ MBR，メモリアドレスレジスタ MAR にロードさ れる.
    （$Load_{memory}$=1 のとき MBR←M[MAR]，$Load_{MBR}$=1 のとき MBR← $データ_{in}$，$Load_{MAR}$=1 のとき MAR← $アドレス_{in}$）



図A・2　メモリ部の回路図



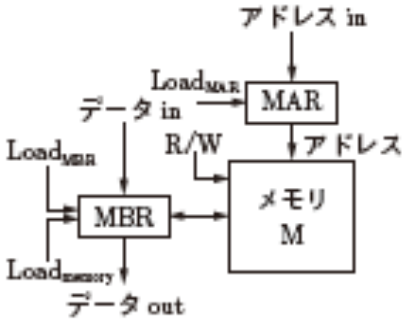
図A・3　プログラムカウンタの回路図



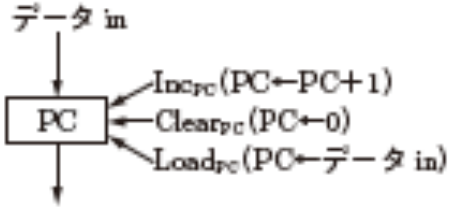
図A・4　算術論理演算装置（ALU）の回路図

利用するIC 部品

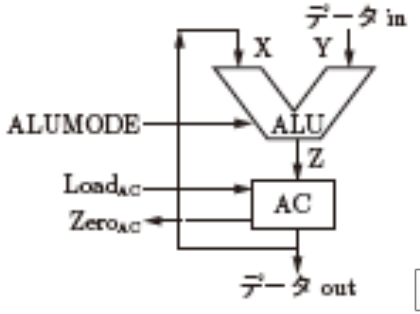
- 上記の四つの命令を実行する CPU を次のような機能をもつメモリ（図A·2），プログラムカウンタ（PC）（図A·3），算術論理演算装置（Arithmetic Logic Unit:ALU）（図A·4），命令レジスタ（Instruction Register: IR）などの部品を用いて実現する.
  - プログラムカウンタPC（図A·3）は，Inc_{PC}, $Clear_{PC}$, $Load_{PC}$ の三つの制御信 号をそれぞれ 1 にすることにより，図A·3 の各制御信号の後ろの ( ) に記載の動作を行う.
  - 例えば，制御信号 Inc_{PC} の値を 1 にすると，プログラムカウンタ PC の 値が 1 増加する． (PC←PC+1)
  - $Clear_{PC}$ 信号はプログラムカウンタ PC の値を 0 にする信号である． (PC←0)
  - $Load_{PC}$ 信号は外部から与えられる“データ$_{in}$” をプログラムカウンタ PC にセットする制御信号である． (PC←データ$_{in}$)



図A・2 メモリ部の回路図



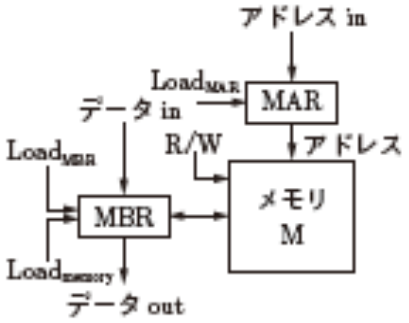
図A・3 プログラムカウンタの回路図



図A・4 算術論理演算装置（ALU）の回路図

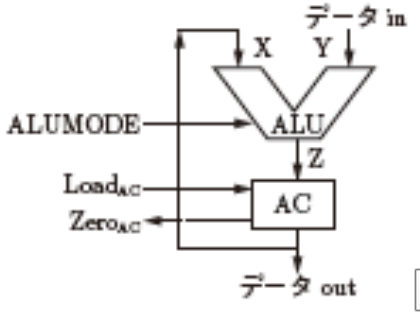
利用するIC 部品

- 上記の四つの命令を実行する CPU を次のような機能をもつメモリ（図A·2），プログラムカウンタ（PC）（図A·3），算術論理演算装置（Arithmetic Logic Unit:ALU）（図A·4），命令レジスタ（Instruction Register: IR）などの部品を用いて実現する.
  - 図A·4 の算術論理演算装置（ALU）は，ALUMODE が ALUMODE = ADD のとき Z←X+Y（すなわちZ←AC+Y）
  - ALUMODE = PASS のとき Z←Y の機能をもつ.
  - アキュムレータ AC は $Load_{AC}$ 信号を 1 にすることにより，Z の 値をロードする.（AC←AC+YまたはAC←Y ）
  - また，AC の値が 0 のときのみ，AC からの出力 $Zero_{AC}$ 信号が 1 になる.
  - この例では ALU の演算として加算のみを考えているが，論理和，論理積，否定，シフト，減算等の機能をもつ ALU を想定してもよい．その場合，ALUMODE を演算の数だけ用意し，その値によって ALU の演算の内容を変化させればよい.



図A・2 メモリ部の回路図

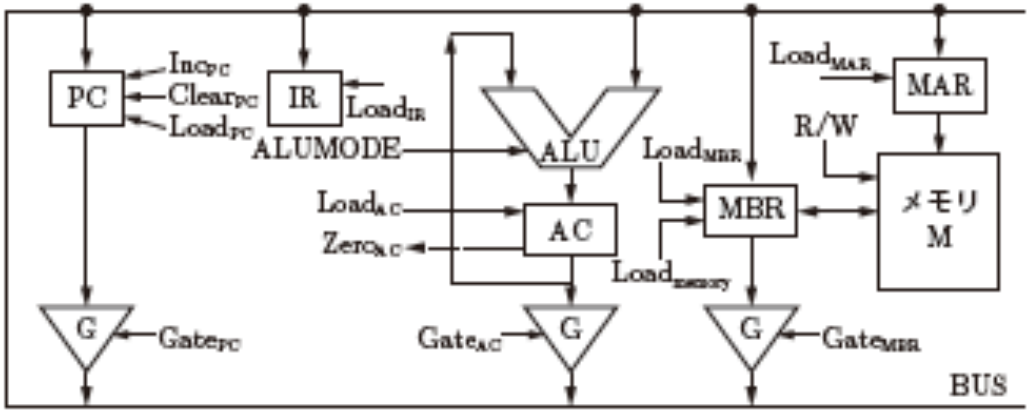
図A・3 プログラムカウンタの回路図



図A・4 算術論理演算装置（ALU）の回路図

CPU 回路の構成

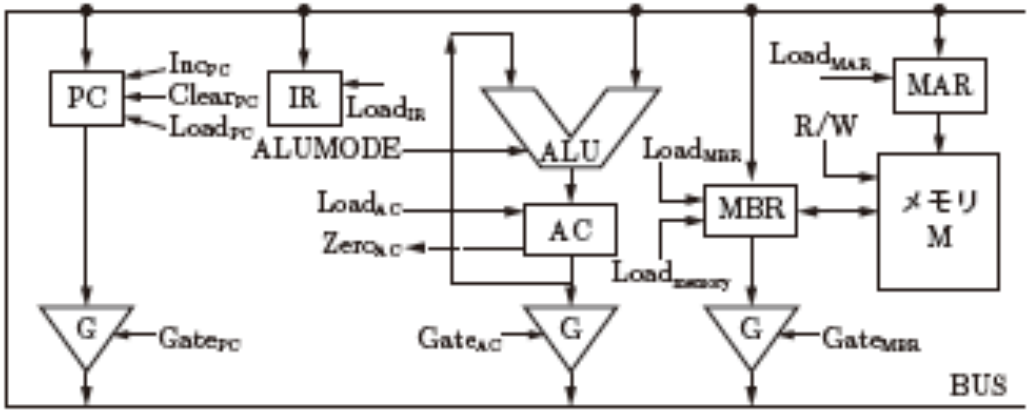
- 以下では CPU回路を 図A·5 のように，メモリ (M)，プログラムカウンタ(PC)，算術論理演算装置(ALU)，レジスタ(MBR, MAR, AC)，命令レジスタ(IR)，三つのゲート回路(G)を一つのバス(BUS)でつないで実現する.
- 各レジスタ間の転送は送信側レジスタ(PC, AC, MBR)のGate 信号($Gate_{PC}$, $Gate_{AC}$, $Gate_{MBR}$) のいずれか一つの信号をオンにし，受信側レジスタ(PC, IR, AC, MBR, MAR)のLoad 信号($Load_{PC}$, $Load_{IR}$, $Load_{AC}$, $Load_{MBR}$, $Load_{MAR}$)をオンにすることにより行われる.
- 三つのGate信号($Gate_{PC}$, $Gate_{AC}$, $Gate_{MBR}$)のいずれか 一つの信号をオンにして送信側レジスタ(PC, AC, MBR)のうちの一つのレジス タの値をバス(BUS)に送出する仕組みはマルチプレクサを用いることで実現可能である.



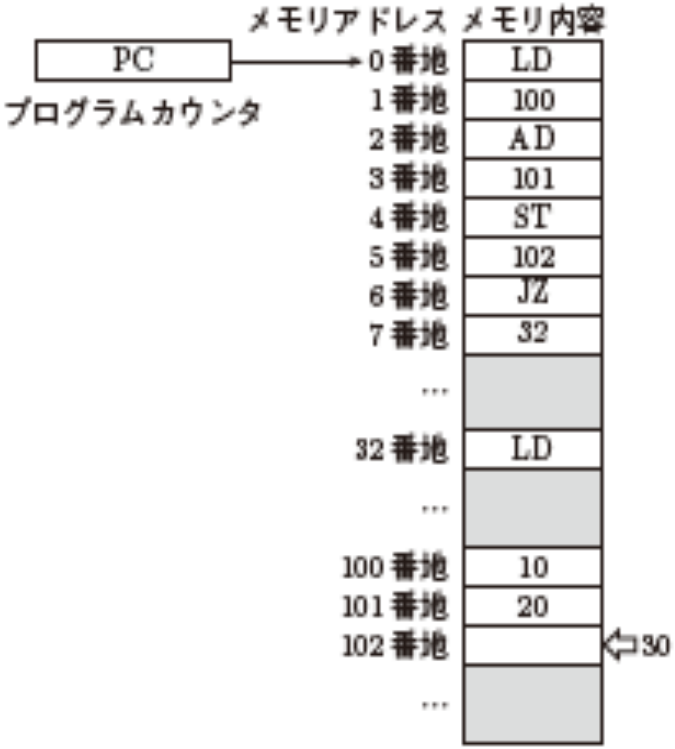
図A・5 BUS を用いた CPU の回路図

プログラムの実行
～フェッチサイクルと実行サイクル～

- 図A·1 のようにメモリに格納された機械語プログラムを実行するには，まずプログラムカウンタ PC の指すメモリの番地から命令を読み出し，命令レジスタ IR にその内容を転送し，その命令がどのような命令であるかを解析（デコード）し， PC+1 番地に書かれたオペランドの内容を読み出すために PC の値を 1 増やす必 要がある．これらの作業をフェッチサイクルと呼ぶ.
- フェッチサイクルが終了すると，命令レジスタ IR の内容に従ってそれぞれの命令の機能に記載された動作列を実行する必要がある．これらの作業を実行サイクルと呼ぶ.



図A・5 BUS を用いた CPU の回路図



図A・1 メモリに格納された機械語プログラム（アセ

プログラムの実行
～フェッチサイクル～

フェッチサイクルの操作列

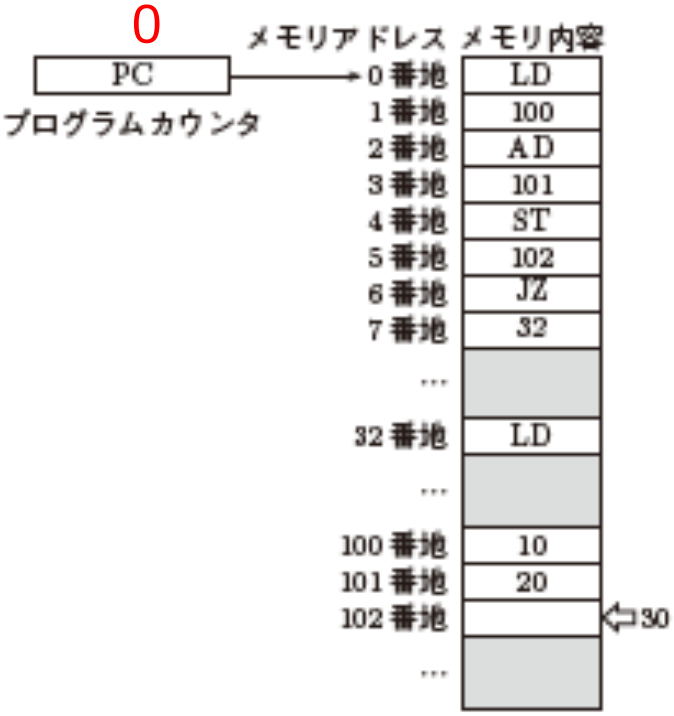
- メモリーの偶数番地に格納されている命令を命令レジスタ IR に転送する.
- フェッチサイクルでは，次の (t_0) から (t_3) の四つの操作を順に実行することで，命令レジスタ IR に命令の名前が格納され，プログラムカウンタ PC がその命令のオペランドを読み出せるようになる.

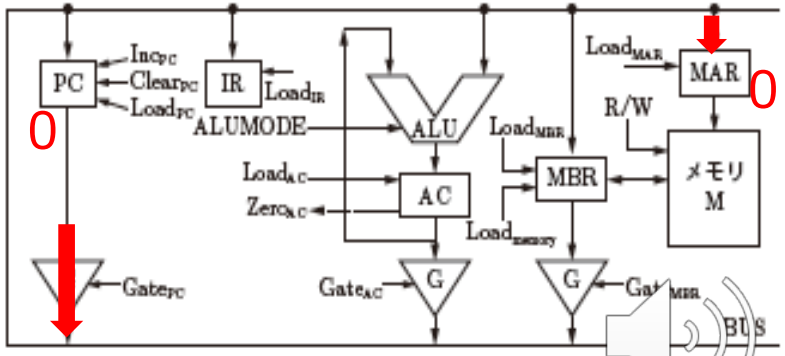
(t_0) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送（MAR←PC）

(t_1) MAR番地の内容 M[MAR] をレジスタ MBR に転送（MBR←M[MAR]）

(t_2) レジスタ MBR の内容を命令レジスタ IR に転送（IR←MBR）

(t_3) プログラムカウンタ PC の値を 1 増やす（PC←PC+1）





図A・5 BUS を用いた CPU の回路図

プログラムの実行
～フェッチサイクル～

フェッチサイクルの操作列

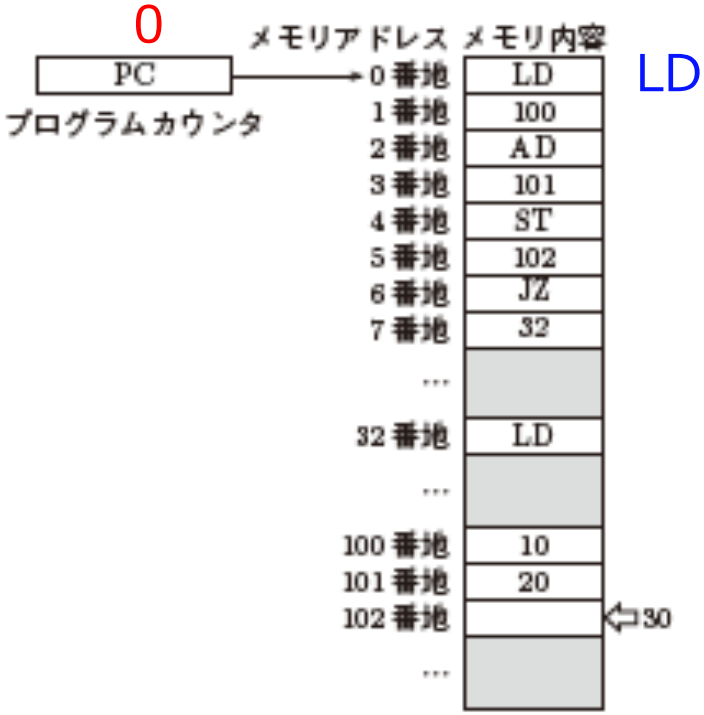
- メモリーの偶数番地に格納されている命令を命令レジスタ IR に転送する.
- フェッチサイクルでは，次の (t_0) から (t_3) の四つの操作を順に実行することで，命令レジスタ IR に命令の名前が格納され，プログラムカウンタ PC がその命令のオペランドを読み出せるようになる.

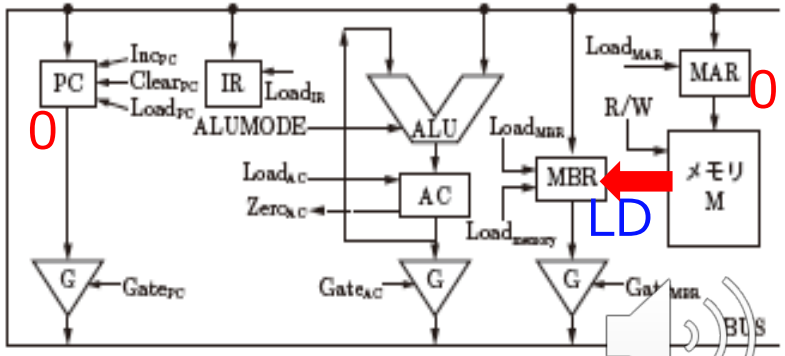
(t_0) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送（MAR←PC） 0

(t_1) MAR番地の内容 M[MAR] をレジスタ MBR に転送（MBR←M[MAR]） LD

(t_2) レジスタ MBR の内容を命令レジスタ IR に転送（IR←MBR）

(t_3) プログラムカウンタ PC の値を 1 増やす（PC←PC+1）





図A・5 BUS を用いた CPU の回路図

プログラムの実行
～フェッチサイクル～

フェッチサイクルの操作列

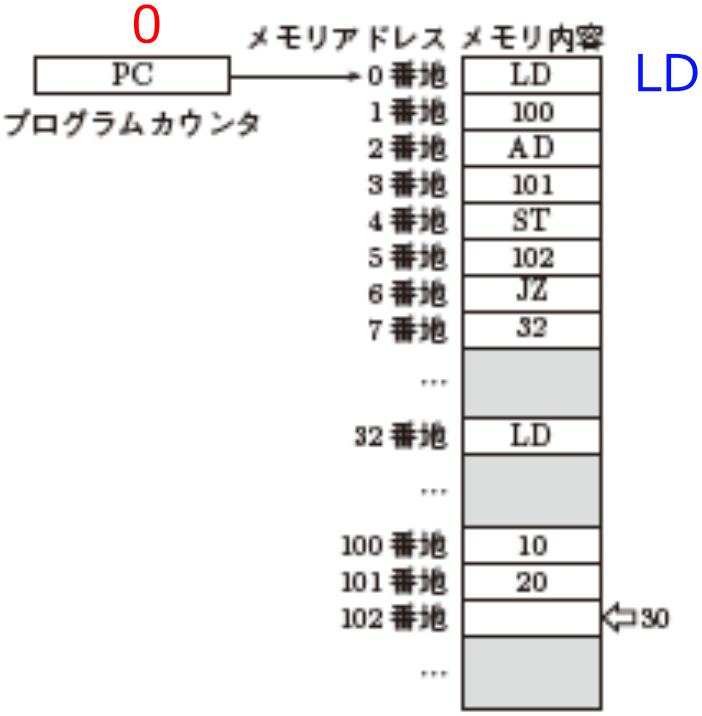
- メモリーの偶数番地に格納されている命令を命令レジスタ IR に転送する.
- フェッチサイクルでは，次の (t_0) から (t_3) の四つの操作を順に実行することで，命令レジスタ IR に命令の名前が格納され，プログラムカウンタ PC がその命令のオペランドを読み出せるようになる.

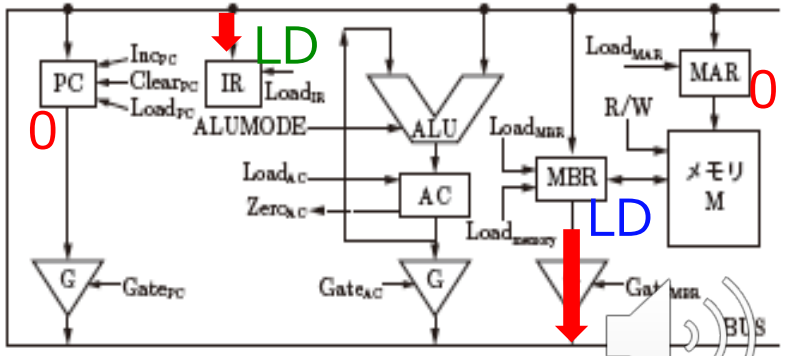
(t_0) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送（MAR←PC） 0

(t_1) MAR番地の内容 M[MAR] をレジスタ MBR に転送（MBR←M[MAR]） LD

(t_2) レジスタ MBR の内容を命令レジスタ IR に転送（IR←MBR） LD

(t_3) プログラムカウンタ PC の値を 1 増やす（PC←PC+1）





図A・5 BUS を用いた CPU の回路図

プログラムの実行
～フェッチサイクル～

フェッチサイクルの操作列

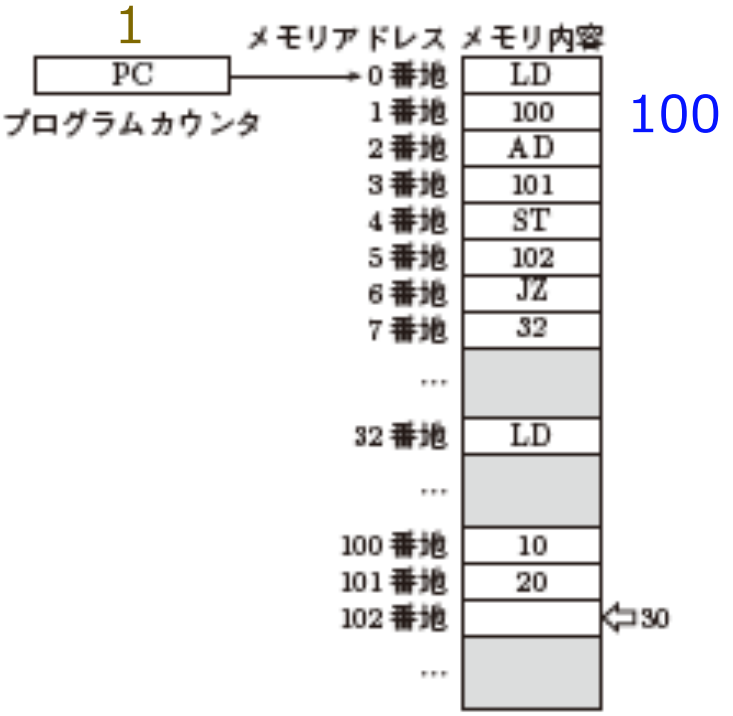
- メモリーの偶数番地に格納されている命令を命令レジスタ IR に転送する.
- フェッチサイクルでは，次の (t_0) から (t_3) の四つの操作を順に実行することで，命令レジスタ IR に命令の名前が格納され，プログラムカウンタ PC がその命令のオペランドを読み出せるようになる.

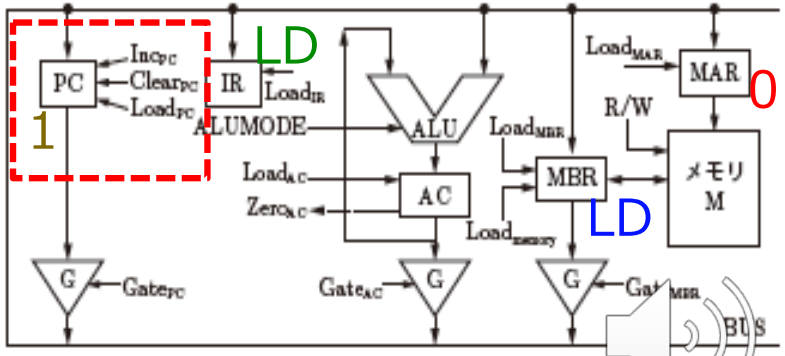
(t_0) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送（MAR←PC） 0

(t_1) MAR番地の内容 M[MAR] をレジスタ MBR に転送（MBR←M[MAR]） LD

(t_2) レジスタ MBR の内容を命令レジスタ IR に転送（IR←MBR） LD

(t_3) プログラムカウンタ PC の値を 1 増やす（PC←PC+1） 1





図A・5 BUS を用いた CPU の回路図

プログラムの実行
～ロード（LD）命令の実行サイクル～

実行サイクルの操作列

- ロード（LD）命令は，そのオペランドに記載の番地（adr 番地）のデータをレジスタ AC に転送する命令である.

(t_4) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送
（MAR←PC） 1

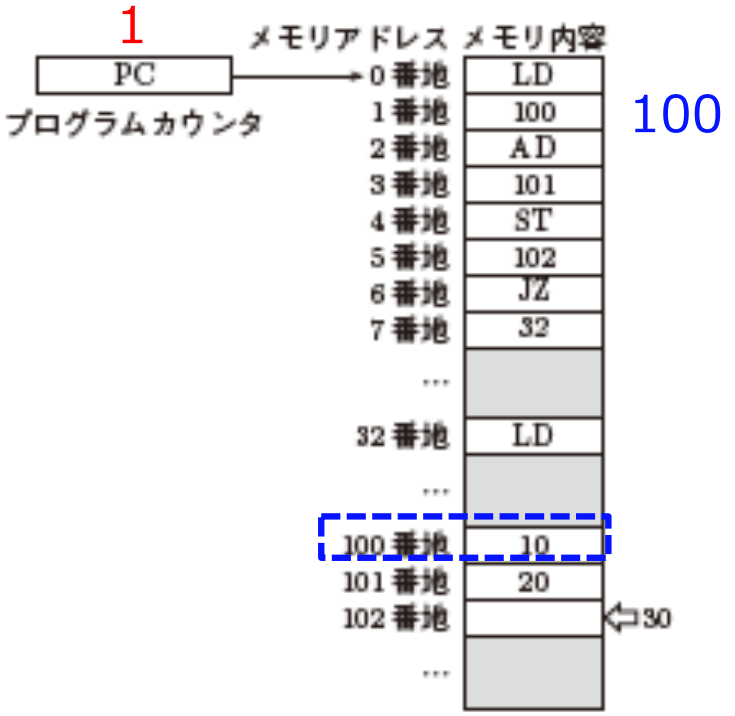
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの adr 番地を取得

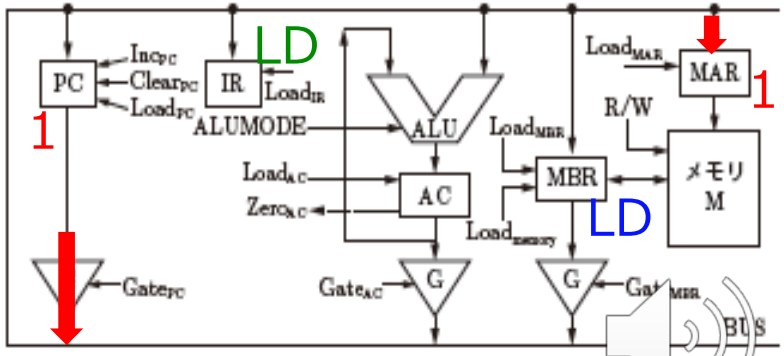
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR）

(t_7) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR← M[MAR]）

(t_8) レジスタ MBR の内容をレジスタ AC にロード
（AC←MBR）

(t_9) プログラムカウンタ PC の値を 1 増やす
（PC←PC+1）





図A・5 BUS を用いた CPU の回路図

プログラムの実行
～ロード（LD）命令の実行サイクル～

実行サイクルの操作列

- ロード（LD）命令は，そのオペランドに記載の番地（adr 番地）のデータをレジスタ AC に転送する命令である.

(t_4) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送
（MAR←PC）

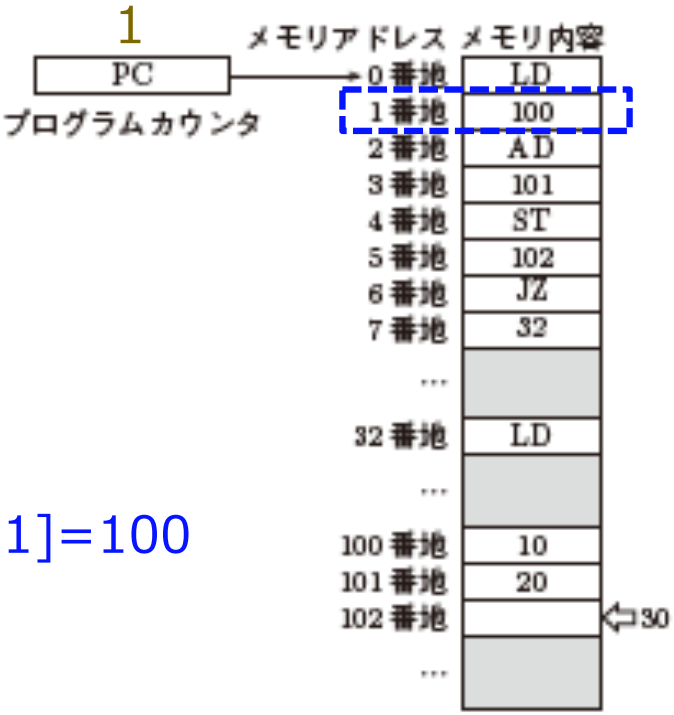
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）オペランドの adr 番地を取得　M[1]=100

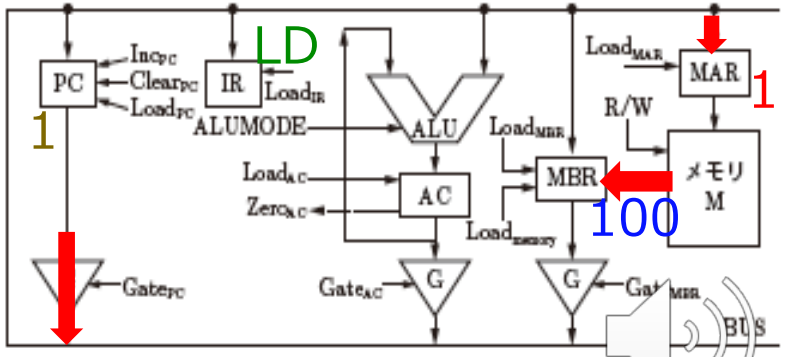
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR）

(t_7) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR← M[MAR]）

(t_8) レジスタ MBR の内容をレジスタ AC にロード
（AC←MBR）

(t_9) プログラムカウンタ PC の値を 1 増やす
（PC←PC+1）





図A・5　BUS を用いた CPU の回路図

プログラムの実行
～ロード（LD）命令の実行サイクル～

実行サイクルの操作列

- ロード（LD）命令は，そのオペランドに記載の番地（adr 番地）のデータをレジスタ AC に転送する命令である.

(t_4) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送
（MAR←PC）

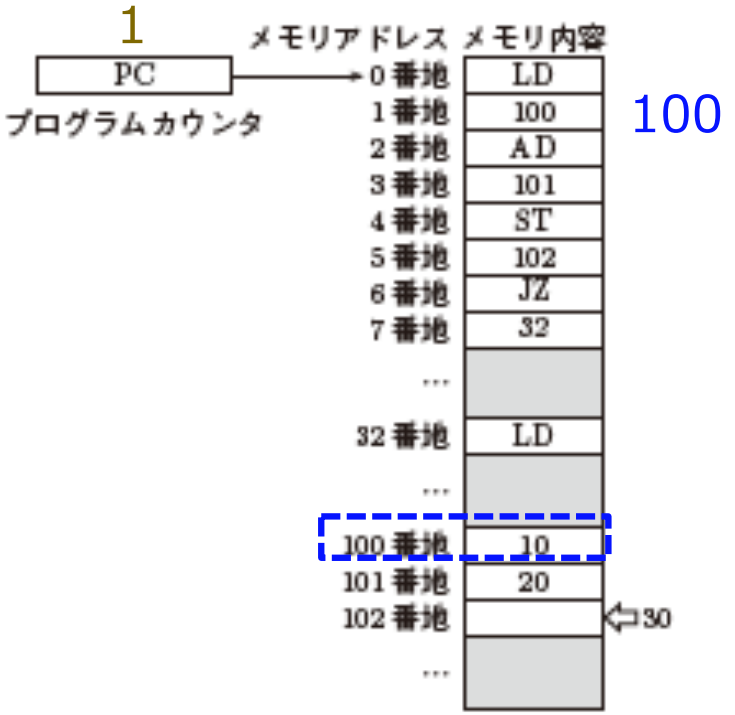
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの adr 番地を取得

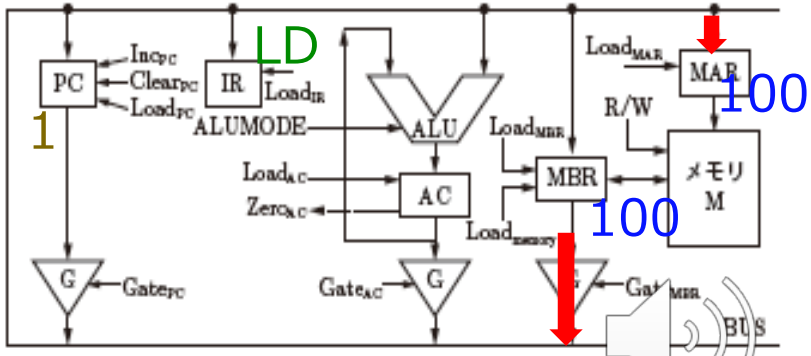
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR） 100

(t_7) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR← M[MAR]）

(t_8) レジスタ MBR の内容をレジスタ AC にロード
（AC←MBR）

(t_9) プログラムカウンタ PC の値を 1 増やす
（PC←PC+1）





図A・5 BUS を用いた CPU の回路図

プログラムの実行
～ロード（LD）命令の実行サイクル～

実行サイクルの操作列

- ロード（LD）命令は，そのオペランドに記載の番地（adr 番地）のデータをレジスタ AC に転送する命令である.

(t_4) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送
（MAR←PC）

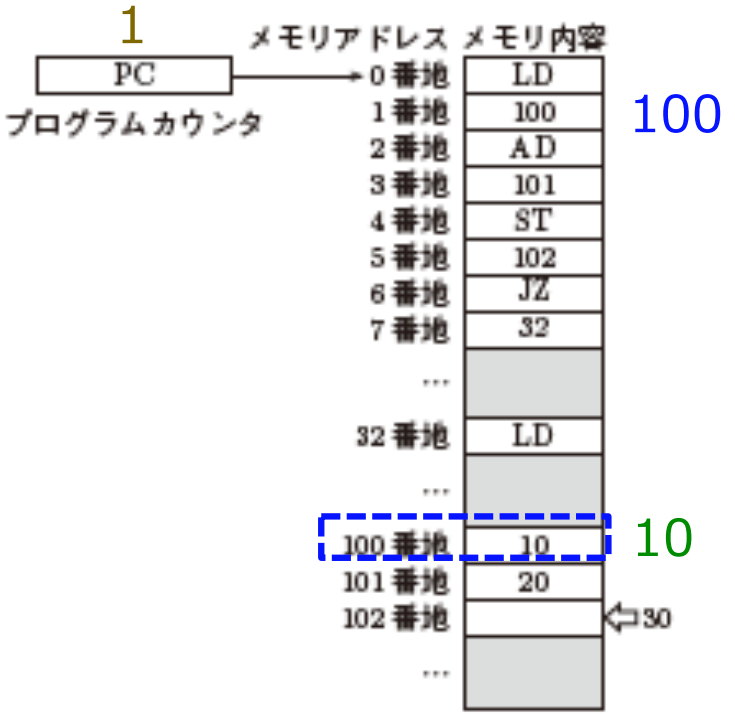
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの adr 番地を取得

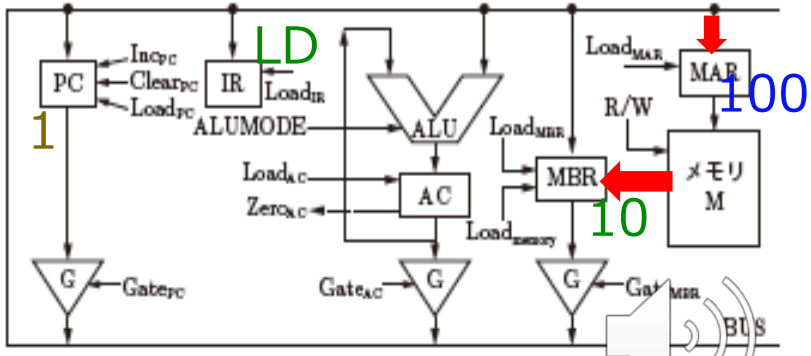
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR） 100

(t_7) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR← M[MAR]） 10

(t_8) レジスタ MBR の内容をレジスタ AC にロード
（AC←MBR）

(t_9) プログラムカウンタ PC の値を 1 増やす
（PC←PC+1）





図A・5 BUS を用いた CPU の回路図

プログラムの実行
～ロード（LD）命令の実行サイクル～

実行サイクルの操作列

- ロード（LD）命令は，そのオペランドに記載の番地（adr 番地）のデータをレジスタ AC に転送する命令である.

(t_4) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送
（MAR←PC）

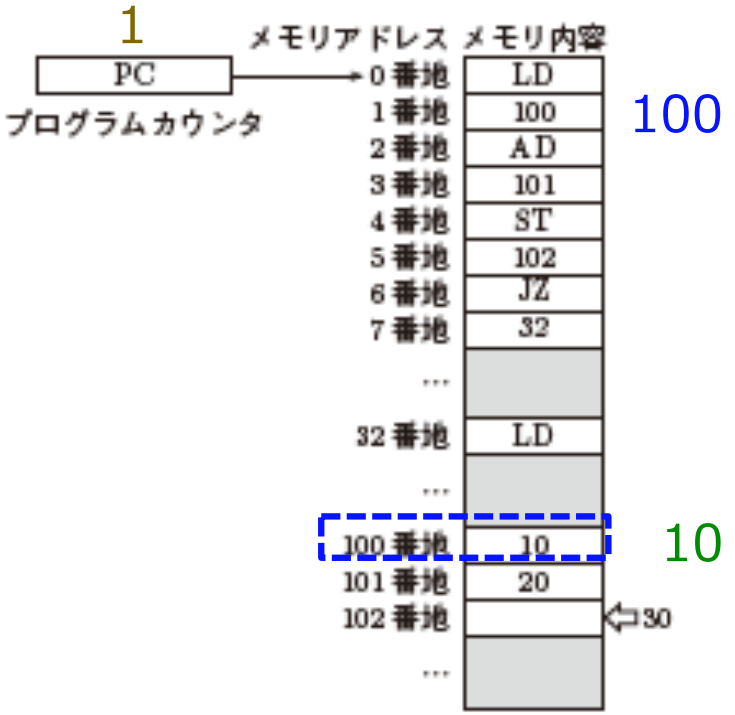
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの adr 番地を取得

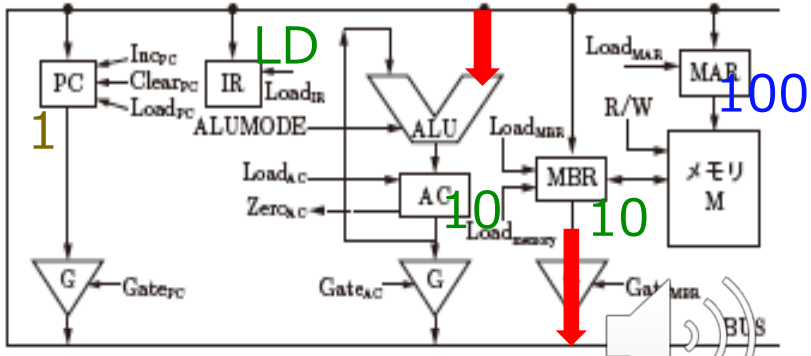
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR） 100

(t_7) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR← M[MAR]） 10

(t_8) レジスタ MBR の内容をレジスタ AC にロード
（AC←MBR） 10

(t_9) プログラムカウンタ PC の値を 1 増やす
（PC←PC+1）





図A・5 BUS を用いた CPU の回路図

プログラムの実行
～ロード（LD）命令の実行サイクル～

実行サイクルの操作列

- ロード（LD）命令は，そのオペランドに記載の番地（adr 番地）のデータをレジスタ AC に転送する命令である.

(t_4) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送
（MAR←PC）

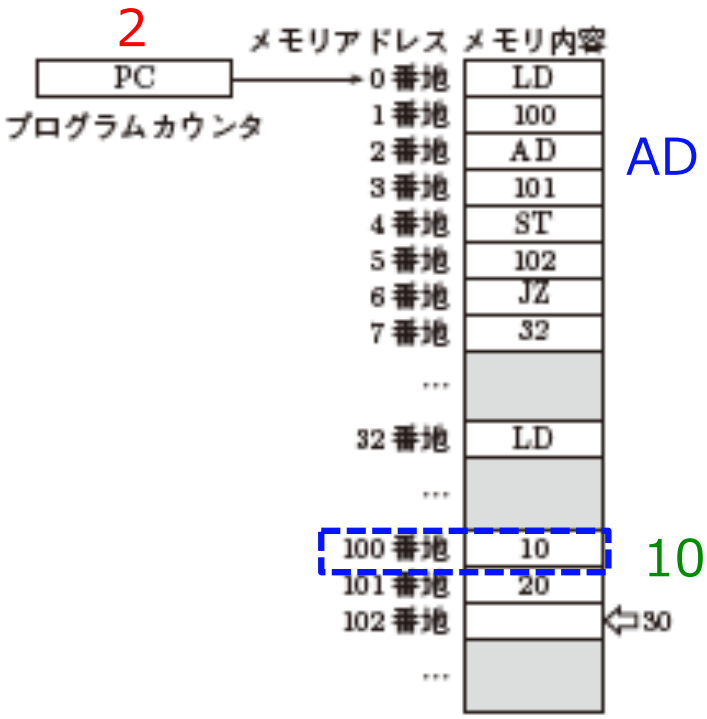
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの adr 番地を取得

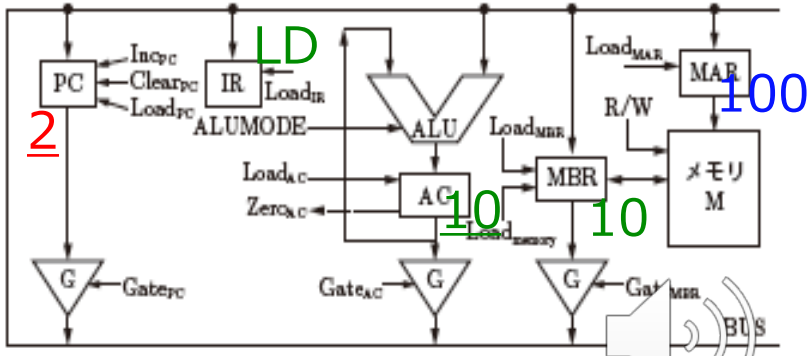
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR） 100

(t_7) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR← M[MAR]） 10

(t_8) レジスタ MBR の内容をレジスタ AC にロード
（AC←MBR） 10

(t_9) プログラムカウンタ PC の値を 1 増やす
（PC←PC+1） 2





図A・5 BUS を用いた CPU の回路図

プログラムの実行
～ロード（LD）命令の実行サイクル～

実行サイクルの操作列

- ロード（LD）命令は，そのオペランドに記載の番地（adr 番地）のデータをレジスタ AC に転送する命令である.

(t_4) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送
（MAR←PC）

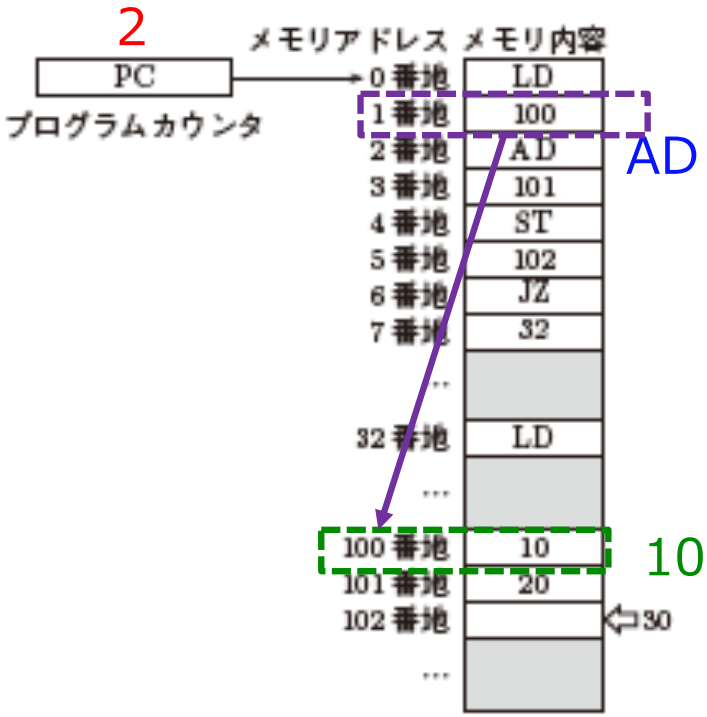
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの adr 番地を取得

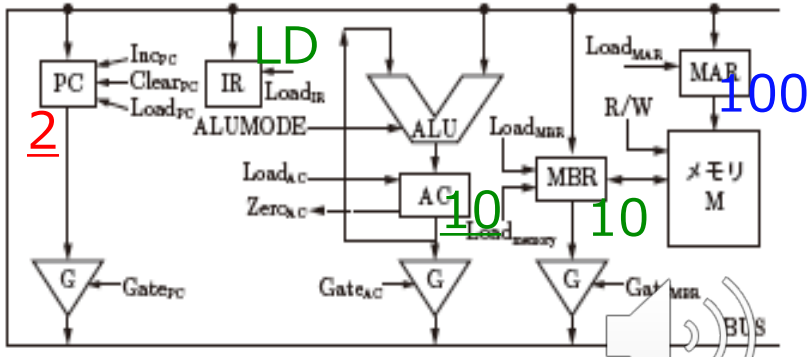
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR） 100

(t_7) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR← M[MAR]） 10

(t_8) レジスタ MBR の内容をレジスタ AC にロード
（AC←MBR） 10

(t_9) プログラムカウンタ PC の値を 1 増やす
（PC←PC+1） 2





図A・5 BUS を用いた CPU の回路図

プログラムの実行
～フェッチサイクル～

フェッチサイクルの操作列

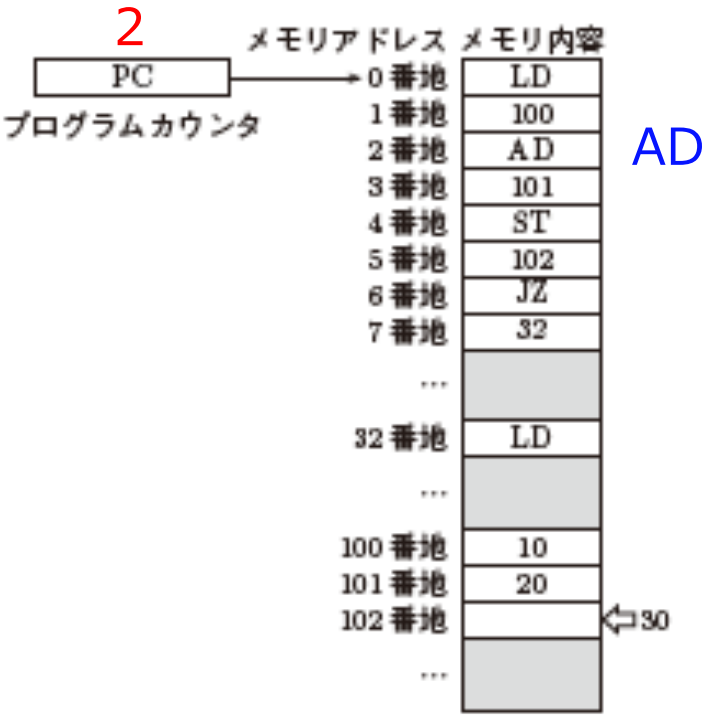
- メモリーの偶数番地に格納されている命令を命令レジスタ IR に転送する.
- フェッチサイクルでは，次の (t_0) から (t_3) の四つの操作を順に実行することで，命令レジスタ IR に命令の名前が格納され，プログラムカウンタ PC がその命令のオペランドを読み出せるようになる.

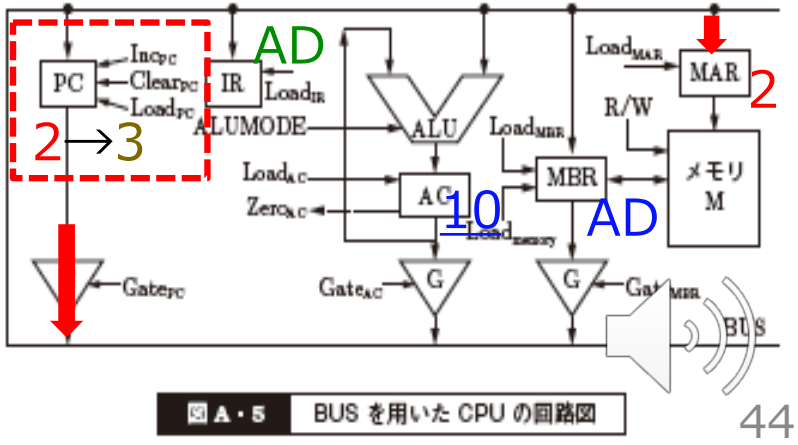
(t_0) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送（MAR←PC） 2

(t_1) MAR番地の内容 M[MAR] をレジスタ MBR に転送（MBR←M[MAR]） AD

(t_2) レジスタ MBR の内容を命令レジスタ IR に転送（IR←MBR） AD

(t_3) プログラムカウンタ PC の値を 1 増やす（PC←PC+1） 3





図A・5 BUS を用いた CPU の回路図

プログラムの実行
～加算（AD）命令の実行サイクル～

実行サイクルの操作列

- 加算（AD）命令は，そのオペランドに記載の番地（adr 番地）のデータをレジスタ AC に加算する命令である.

(t_4) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送
（MAR←PC） 3

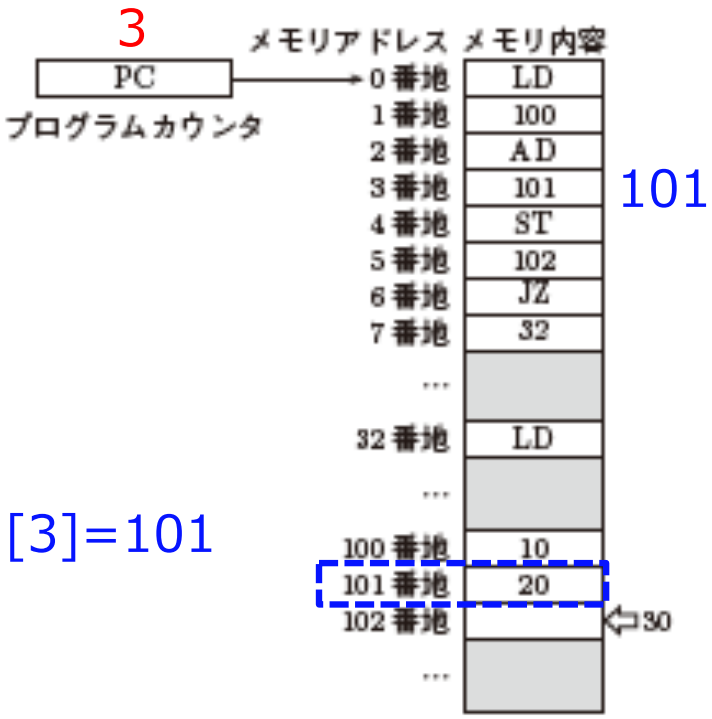
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの adr 番地を取得

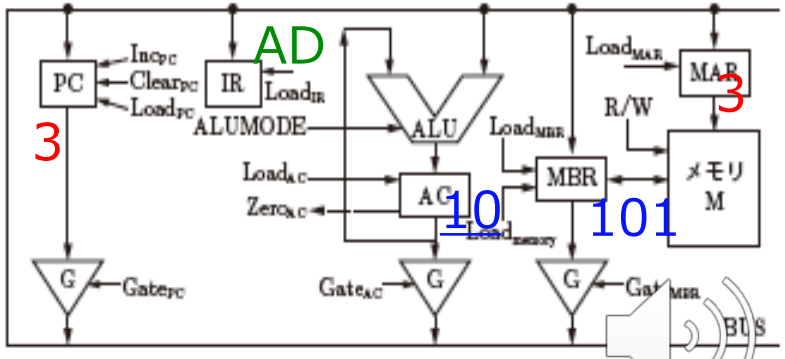
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR）

(t_7) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）

(t_8) レジスタ AC にレジスタ MBR の内容を加算
（AC←AC+MBR）

(t_9) プログラムカウンタ PC の値を 1 増やす
（PC←PC+1）





図A・5 BUS を用いた CPU の回路図

プログラムの実行
～加算（AD）命令の実行サイクル～

実行サイクルの操作列

- 加算（AD）命令は，そのオペランドに記載の番地（adr 番地）のデータをレジスタ AC に加算する命令である.

(t_4) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送
（MAR←PC） 3

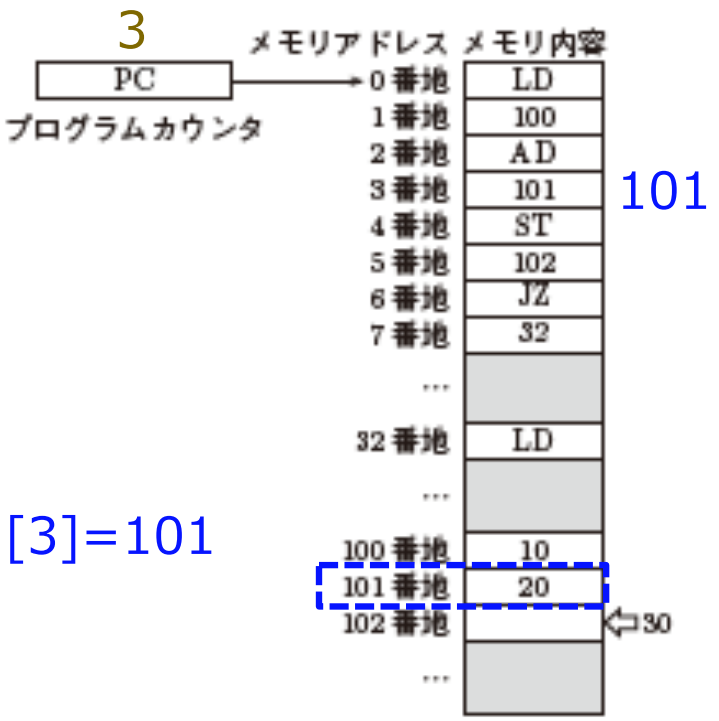
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの adr 番地を取得 M[3]=101

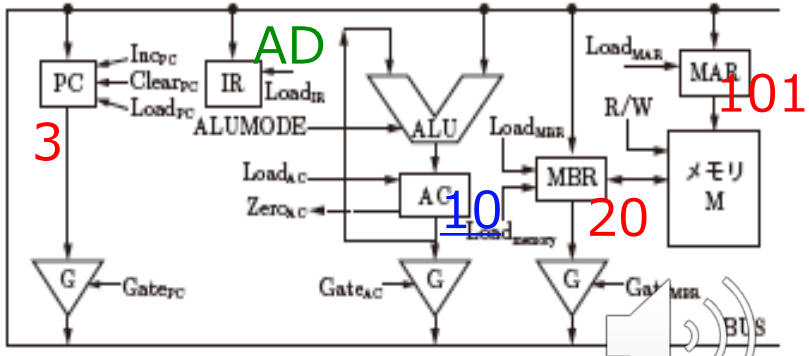
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR） 101

(t_7) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]） M[101]=20

(t_8) レジスタ AC にレジスタ MBR の内容を加算
（AC←AC+MBR）

(t_9) プログラムカウンタ PC の値を 1 増やす
（PC←PC+1）





図A・5 BUS を用いた CPU の回路図

プログラムの実行
～加算（AD）命令の実行サイクル～

実行サイクルの操作列

- 加算（AD）命令は，そのオペランドに記載の番地（adr 番地）のデータをレジスタ AC に加算する命令である.

(t_4) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送
（MAR←PC） 3

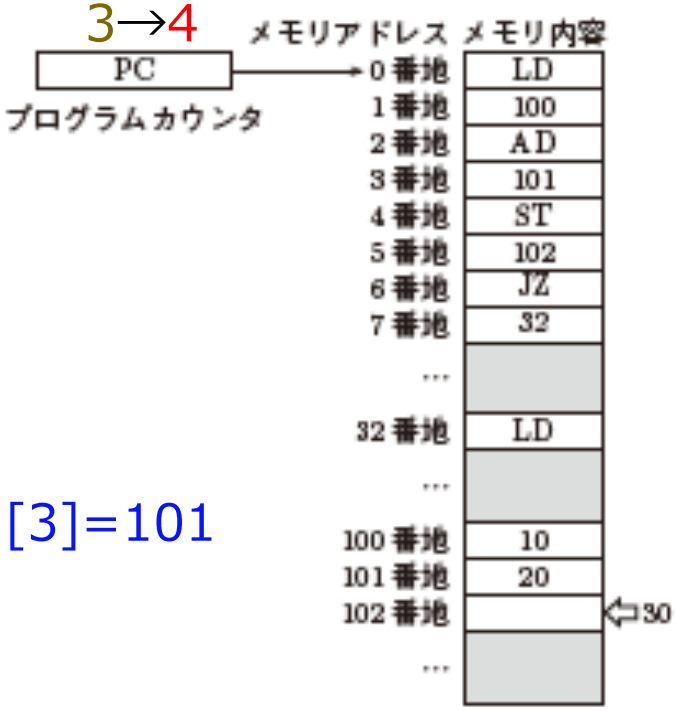
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの adr 番地を取得 M[3]=101

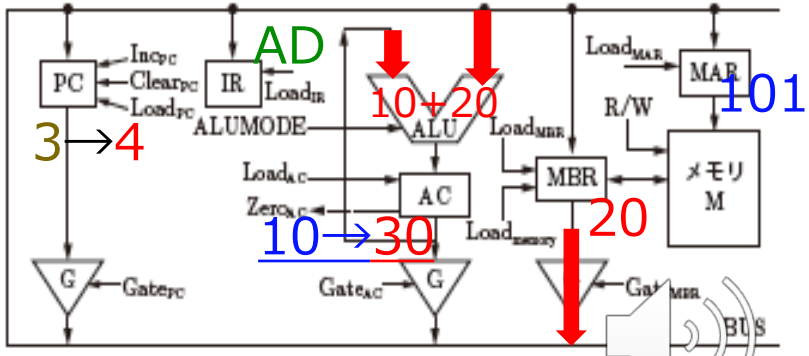
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR） 101

(t_7) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]） M[101]=20

(t_8) レジスタ AC にレジスタ MBR の内容を加算
（AC←AC+MBR） AC←10+20=30

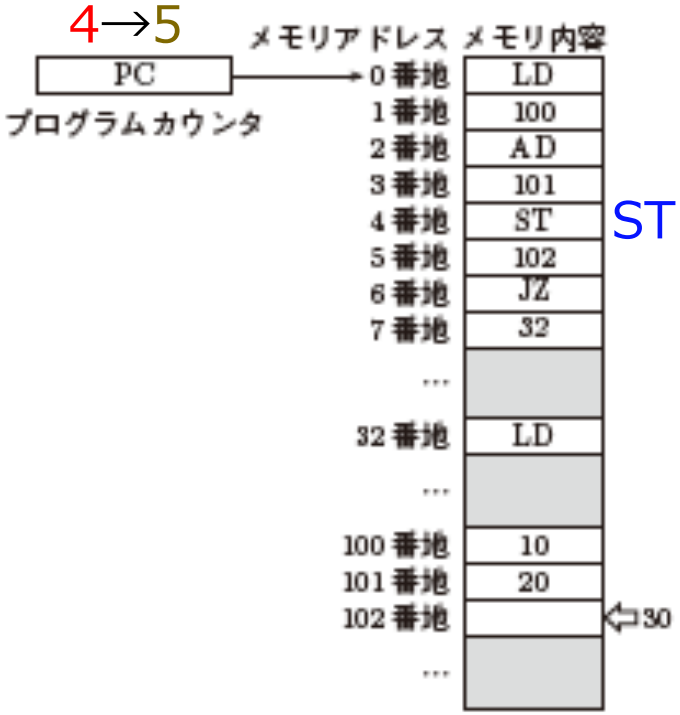
(t_9) プログラムカウンタ PC の値を 1 増やす
（PC←PC+1） 3→4





図A・5 BUS を用いた CPU の回路図

プログラムの実行
～フェッチサイクル～



フェッチサイクルの操作列

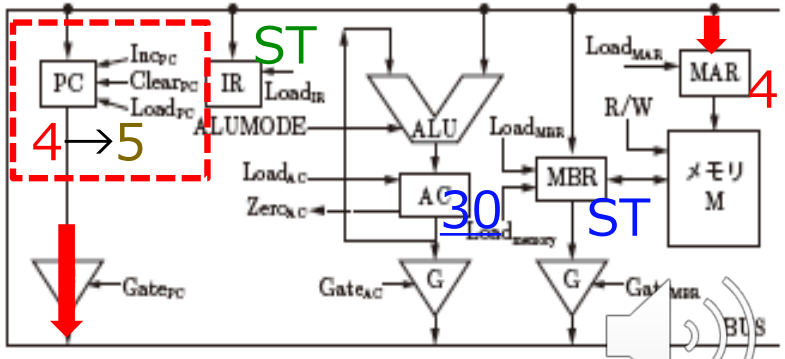
- メモリーの偶数番地に格納されている命令を命令レジスタ IR に転送する.
- フェッチサイクルでは，次の (t_0) から (t_3) の四つの操作を順に実行することで，命令レジスタ IR に命令の名前が格納され，プログラムカウンタ PC がその命令のオペランドを読み出せるようになる.

(t_0) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送（MAR←PC） 4

(t_1) MAR番地の内容 M[MAR] をレジスタ MBR に転送（MBR←M[MAR]） ST

(t_2) レジスタ MBR の内容を命令レジスタ IR に転送（IR←MBR） ST

(t_3) プログラムカウンタ PC の値を 1 増やす（PC←PC+1） 5



図A・5 BUS を用いた CPU の回路図

プログラムの実行
～ストア（ST）命令の実行サイクル～

実行サイクルの操作列

- ストア（ST）命令は，そのオペランドに記載の番地（adr 番地）に レジスタAC の内容を代入する命令である.

(t_4) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送
（MAR←PC） 5

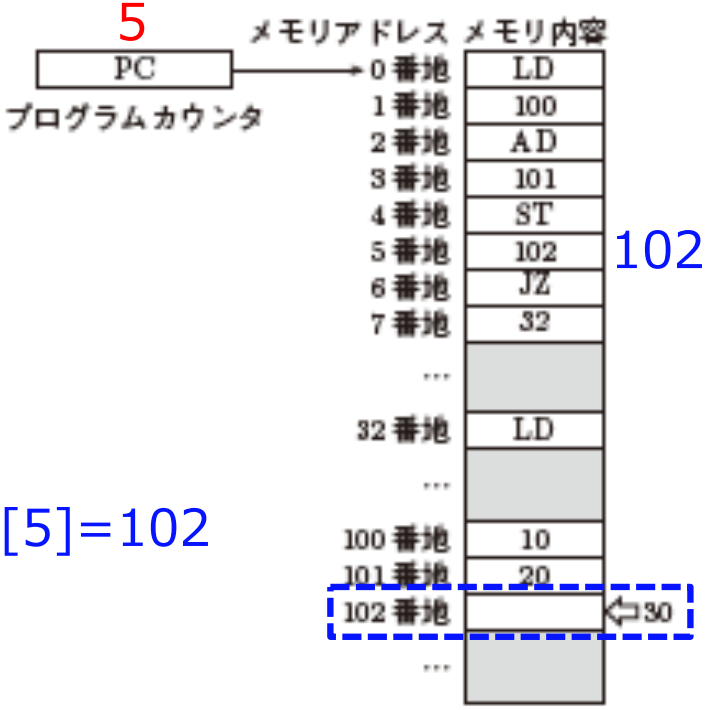
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの adr 番地を取得 M[5]=102

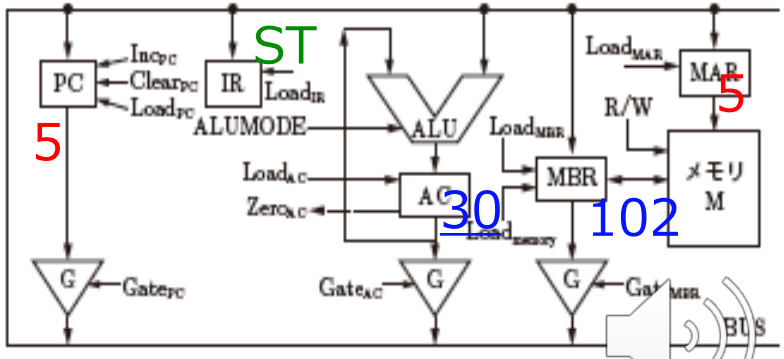
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR）

(t_7) レジスタ AC の内容をレジスタ MBR に転送
（MBR←AC）

(t_8) メモリーM[MAR] にレジスタ MBR の内容を代入
（M[MAR]←MBR）

(t_9) プログラムカウンタ PC の値を 1 増やす
（PC←PC+1）





図A・5 BUS を用いた CPU の回路図

プログラムの実行
～ストア（ST）命令の実行サイクル～

実行サイクルの操作列

- ストア（ST）命令は，そのオペランドに記載の番地（adr 番地）に レジスタAC の内容を代入する命令である.

(t_4) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送
（MAR←PC） 5

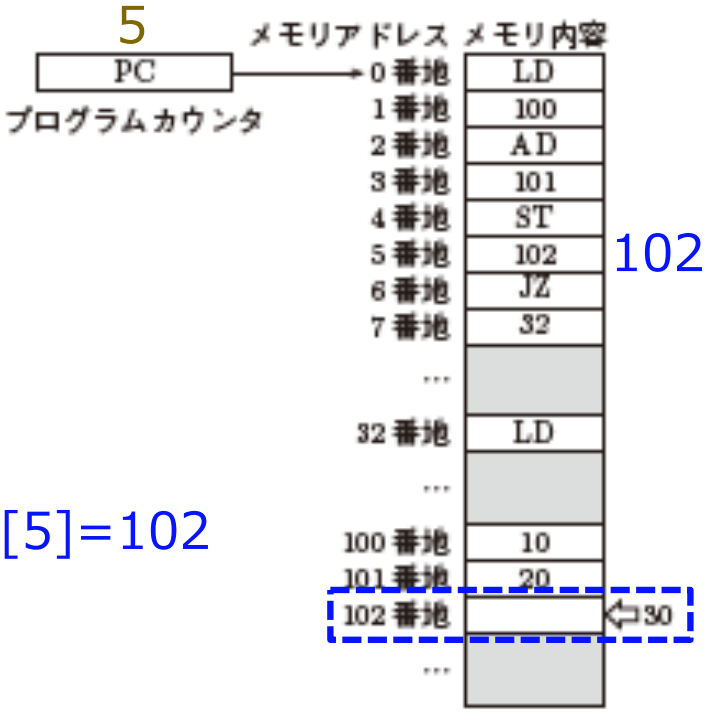
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの adr 番地を取得 M[5]=102

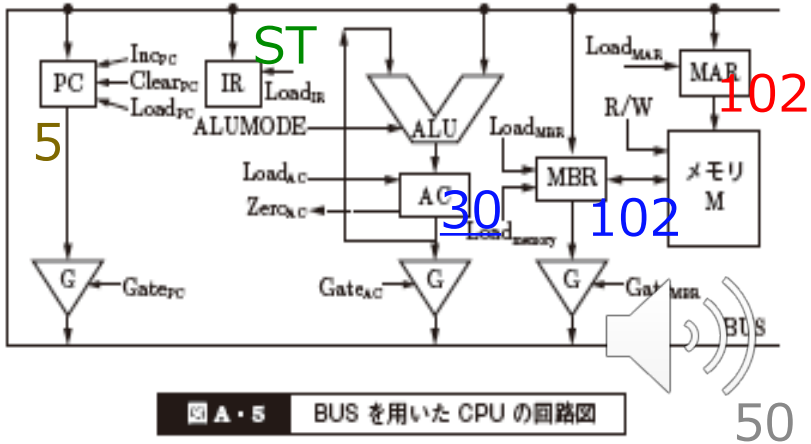
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR） 102

(t_7) レジスタ AC の内容をレジスタ MBR に転送
（MBR←AC）

(t_8) メモリーM[MAR] にレジスタ MBR の内容を代入
（M[MAR]←MBR）

(t_9) プログラムカウンタ PC の値を 1 増やす
（PC←PC+1）





図A・5 BUS を用いた CPU の回路図

プログラムの実行
～ストア（ST）命令の実行サイクル～

実行サイクルの操作列

- ストア（ST）命令は，そのオペランドに記載の番地（adr 番地）に レジスタAC の内容を代入する命令である.

(t_4) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送
（MAR←PC） 5

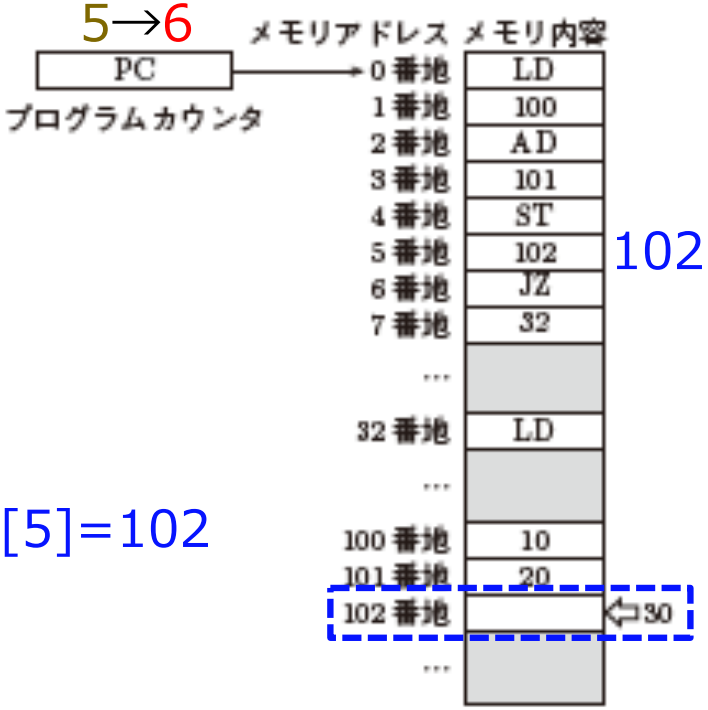
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの adr 番地を取得 M[5]=102

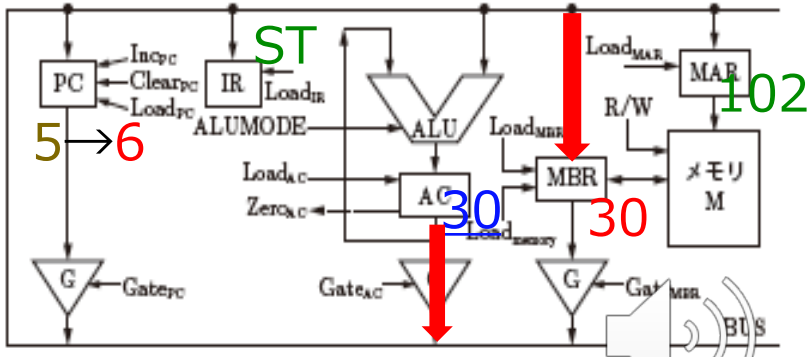
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR） 102

(t_7) レジスタ AC の内容をレジスタ MBR に転送
（MBR←AC） 30

(t_8) メモリーM[MAR] にレジスタ MBR の内容を代入
（M[MAR]←MBR） M[102]←30

(t_9) プログラムカウンタ PC の値を 1 増やす
（PC←PC+1） 5→6





図A・5 BUS を用いた CPU の回路図

プログラムの実行
～フェッチサイクル～

フェッチサイクルの操作列

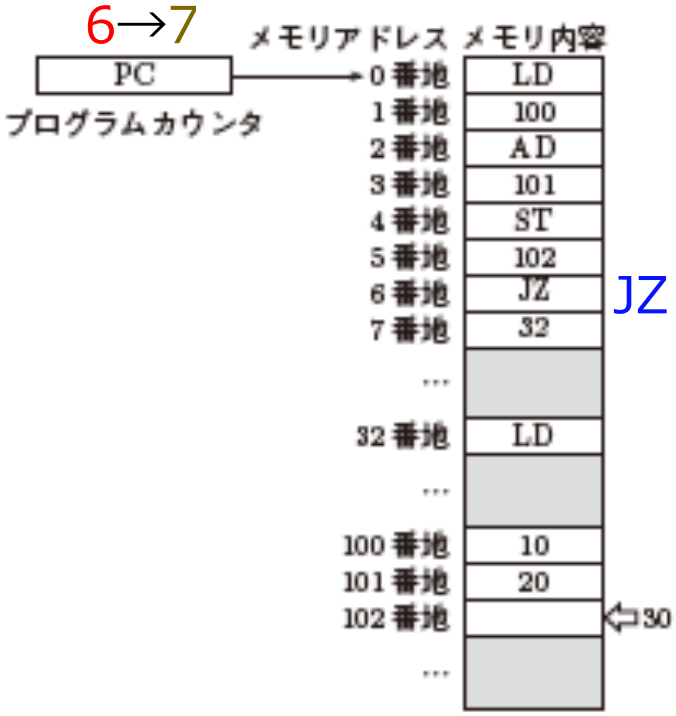
- メモリーの偶数番地に格納されている命令を命令レジスタ IR に転送する.
- フェッチサイクルでは，次の (t_0) から (t_3) の四つの操作を順に実行することで，命令レジスタ IR に命令の名前が格納され，プログラムカウンタ PC がその命令のオペランドを読み出せるようになる.

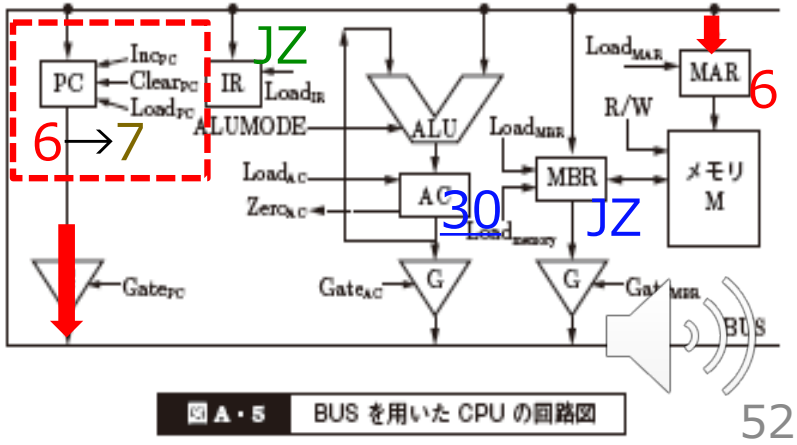
(t_0) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送（MAR←PC） 6

(t_1) MAR番地の内容 M[MAR] をレジスタ MBR に転送（MBR←M[MAR]） JZ

(t_2) レジスタ MBR の内容を命令レジスタ IR に転送（IR←MBR） JZ

(t_3) プログラムカウンタ PC の値を 1 増やす（PC←PC+1） 7





図A・5 BUS を用いた CPU の回路図

プログラムの実行
～条件付ジャンプ（JZ）命令の実行サイクル～

実行サイクルの操作列

- 条件付ジャンプ（JZ）命令は，AC=0 が成り立つときのみプログラムカウンタ PC の値をオペランド B にセットし，そうでなければプログラムカウンタの値を 2 語（inC_4 = 2）増加させる.

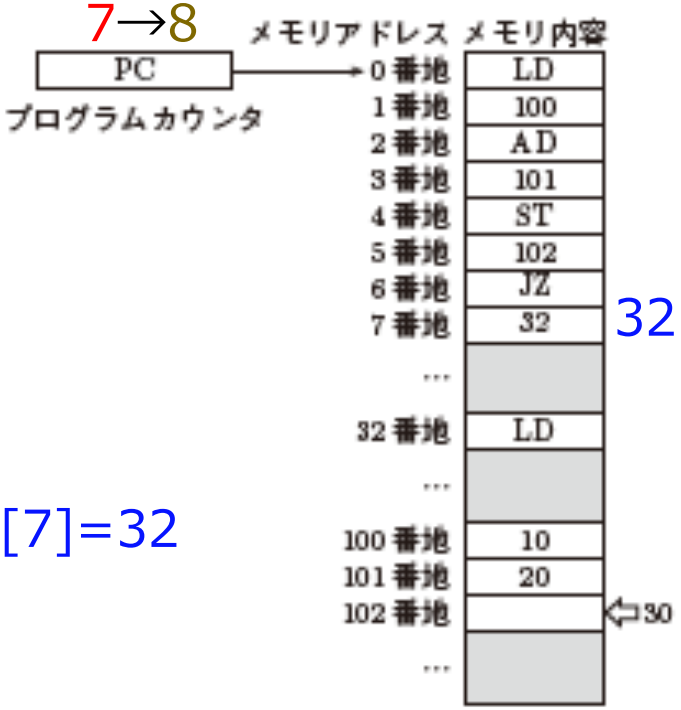
(t_4) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送
（MAR←PC） 7

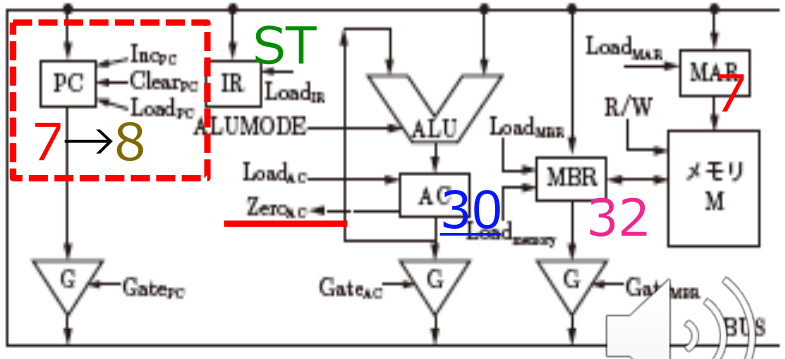
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの B 番地を取得

(t_6) プログラムカウンタ PC の値を 1 増やす
（PC←PC+1） 7→8

(t_7) “Test AC=0” を判定
（Test AC=0?）
No! ($Zero_{AC}$=0) ⇒ フェッチサイクルへ戻る

(t_8) 「Test AC=0?」が Yes のときのみ実行
レジスタ MBR の内容をプログラムカウンタ PC に転送（PC←MBR）

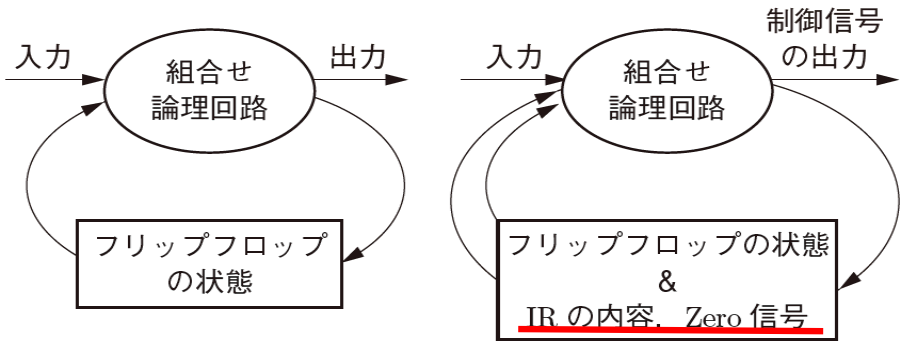




図A・5 BUS を用いた CPU の回路図

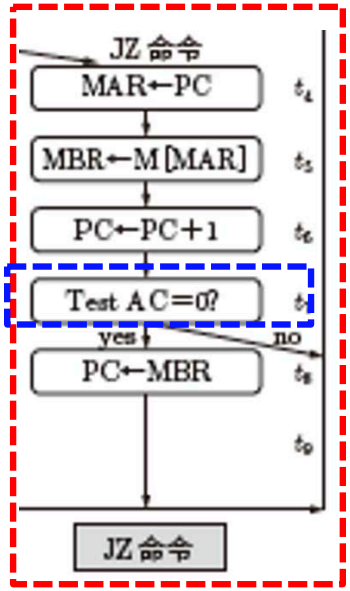
CPU の実現法
～分岐の実現法～

- CPU の動作を同期式順序機械で実現する場合，入力はクロック入力のみであり，JZ 命令のサイクル t_7 の “Test AC=0” を判定する状態については，“Test AC=0” を判定する部分でレジスタ AC の内容に依存して 2 分岐させる必要が生じる.
- 図A·8（b）に示すように，レジスタ AC が 0 であることを表す $Zero_{AC}$ 信号の内容を用いて，次の状態や制御信号の出力を定める組合せ論理回路を実現する.



（a） Moore 型順序機械　　（b） CPU の Moore 型順序機械

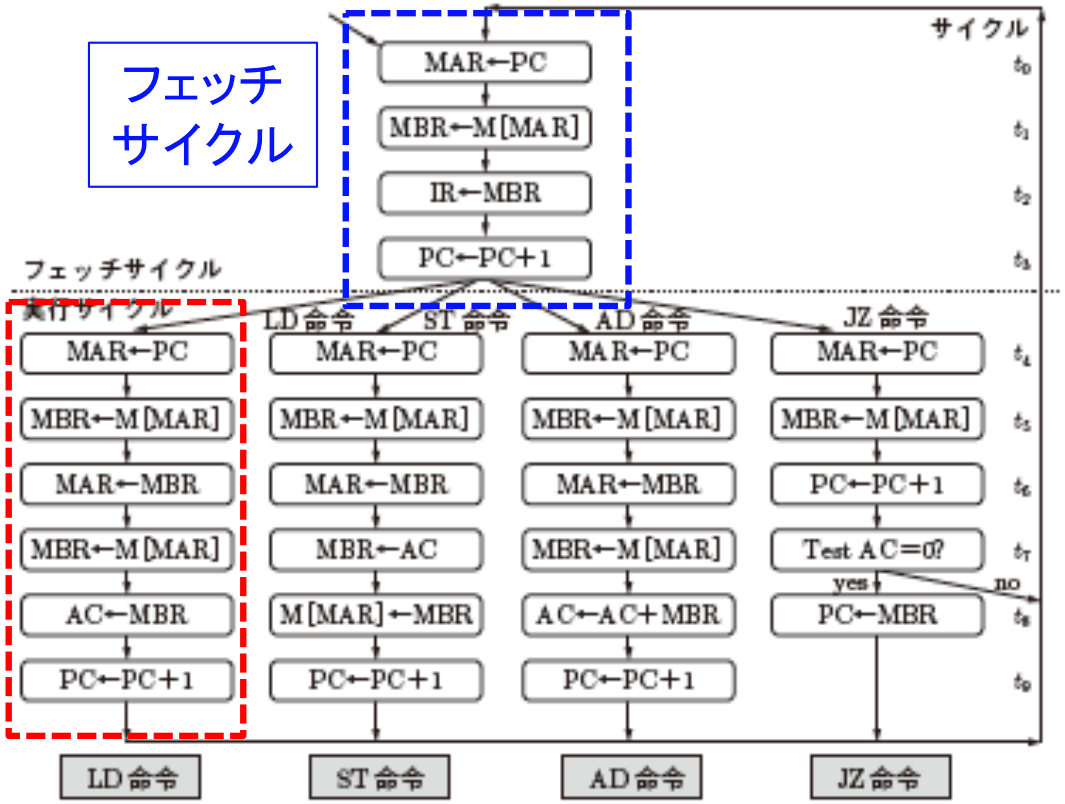
図 A・8 Moore 型順序機械としての実現



2 分岐

ロード（LD）命令の実行サイクルの操作列

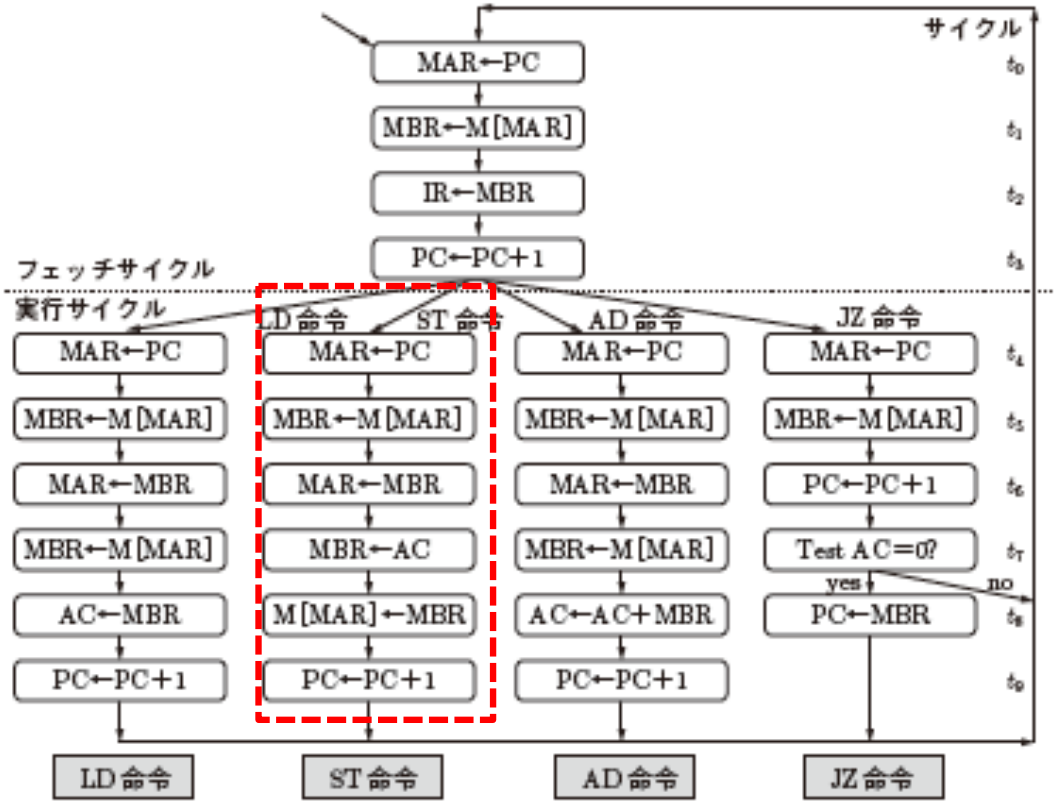
- フェッチサイクルは青字のt_0～t_3のサイクルで実行される．実行サイクルの実施時（t_4サイクル実施時）に４分岐する．
- ロード（LD）命令は，そのオペランドに記載の番地（adr 番地）のデータをレジスタ AC に転送する命令である．



図A・6 制御部の状態遷移図

ストア（ST）命令の実行サイクルの操作列

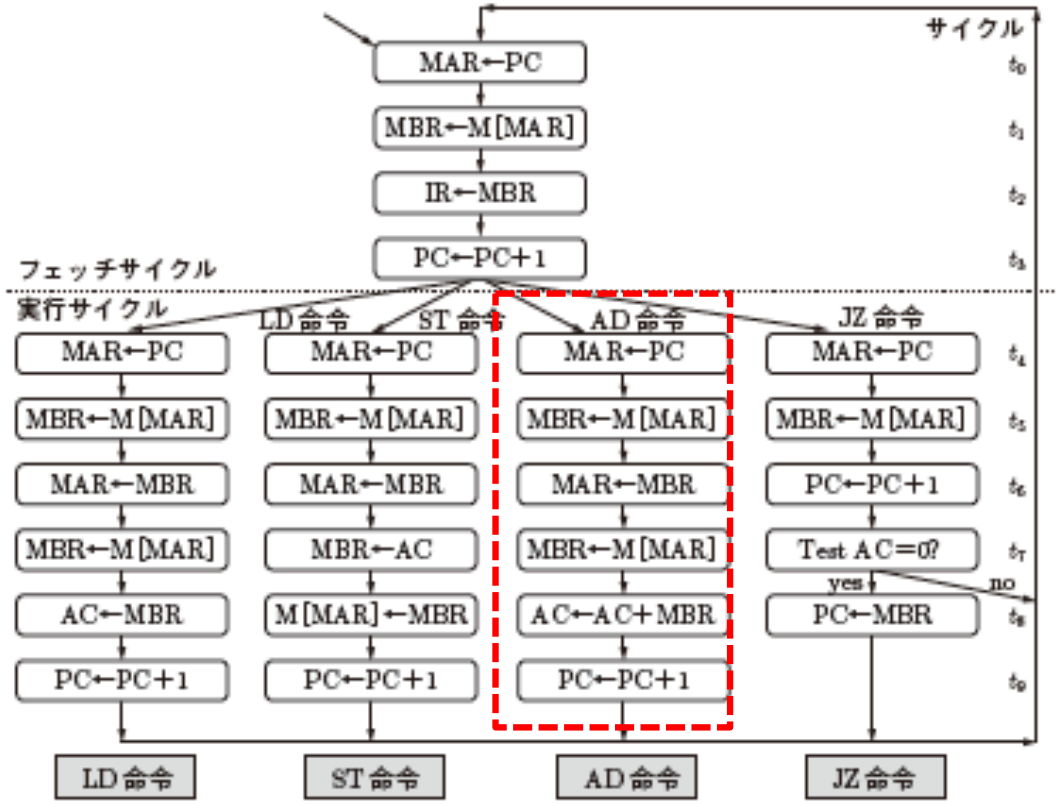
- ストア（ST）命令は，そのオペランドに記載の番地（adr 番地）に レジスタ AC の内容を代入する命令である.



図A・6 制御部の状態遷移図

加算（AD）命令の実行サイクルの操作列

- 加算（AD）命令は，そのオペランドに記載の番地（adr 番地）のデータをレジスタ AC に加算する命令である.



図A・6 制御部の状態遷移図

ジャンプ（JZ）命令の実行サイクルの操作列

- 条件付ジャンプ（JZ）命令は，AC=0 が成り立つときのみプログラムカウンタ PC の値をオペランド B にセットし，そうでなければ次の命令を実行させる（PC←PC+1）．
- “Test AC=0” を判定する部分については，レジスタ AC の内容に依存して 2 分岐させる必要が生じる（フェッチサイクルから実行サイクルに移る際も同様：４分岐）．

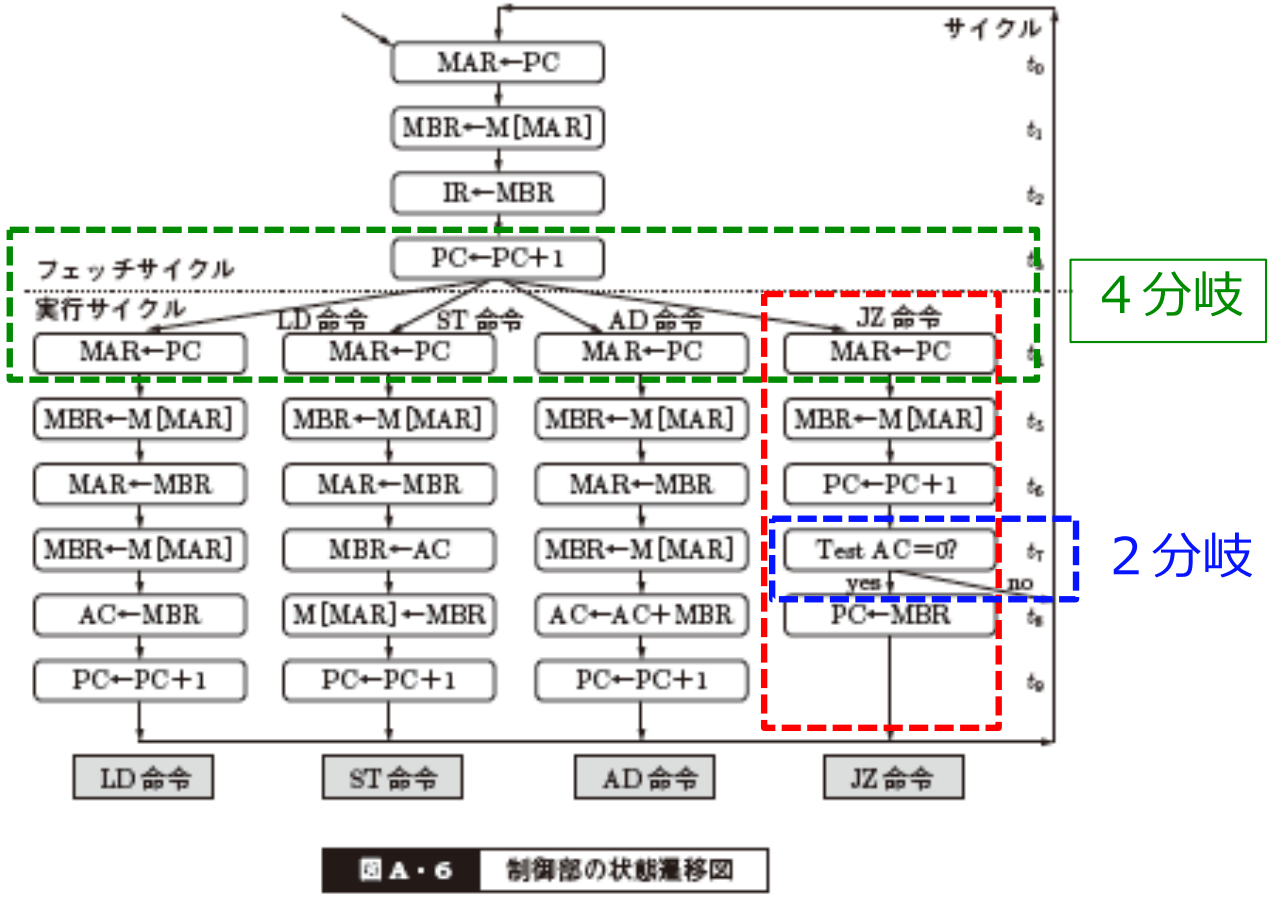
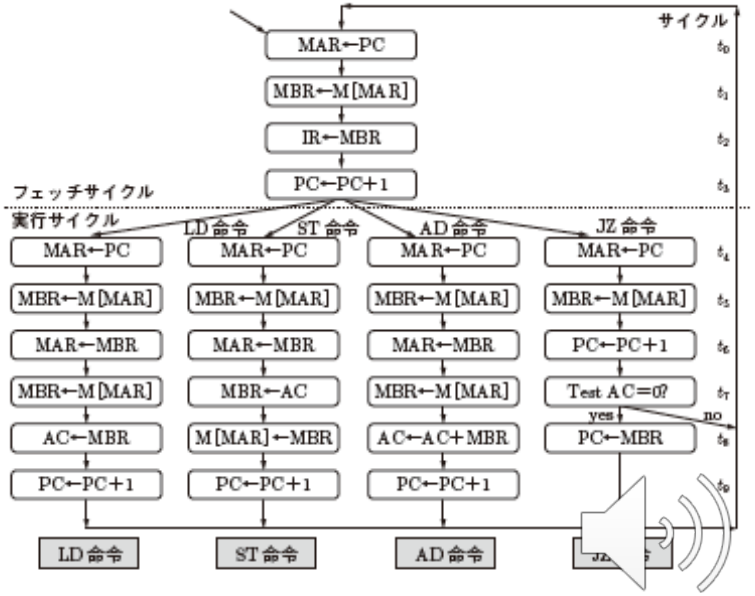


図 A・6 制御部の状態遷移図

CPU の実現法
～Moore 型順序機械としての実現法～

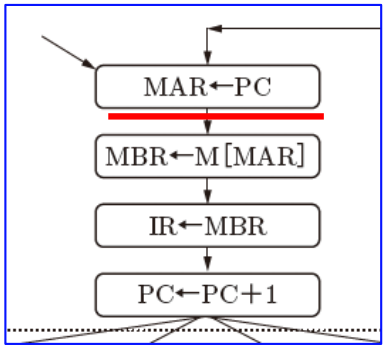
- 上述のフェッチサイクルや実行サイクルの実行順は図A·6 の状態遷移図に記載の Moore 型順序機械で実現できる．図A·6 では，フェッチサイクルが 4 サイクル（t_0,…, t_3），実行サイクルが 6 サイクル（t_4,…,t_9）または 5 サイクル（t_4,…,t_8）で実現され，フェッチサイクルと実行サイクルを順に繰り返し実行することで，機械語のプログラムを順次解釈・実行していく（Moore 型順序機械の各状態の出力については後述する）．
- 上記のフェッチサイクルや実行サイクルは，レジスタ間のデータ転送を行ったり，ALU を用いて加算を行ったり，PC の値を 1 増加する，といった回路上の操作を順次実行することで実現可能である．
- 以下では，このような操作を マイクロ命令 と呼ぶ．ここでは，次の転送，加算，条件判定，カウントアップの四つのマイクロ命令を考える．
  - 転送　　Reg’←Reg”
  - 加算　　AC←AC+Reg”
  - 条件判定　　Test AC=0 ?
  - カウントアップ　　PC←PC+1

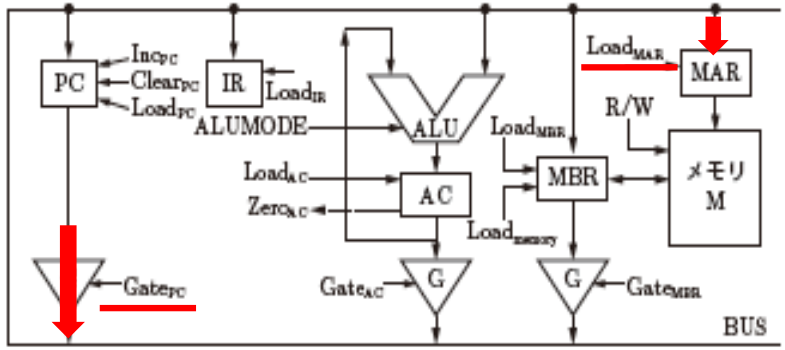


図A・6　制御部の状態遷移図

CPU の実現法
～Moore 型順序機械としての実現法～

- フェッチサイクル，実行サイクルの操作列の各操作を実行するためには，基本的に 図A·7 の各状態でこの図の出力値を出力するようにMoore 型順序機械を実現すればよい.
- 例えば，フェッチサイクルの t_0 は $Gate_{PC} = 1$，$Load_{MAR} = 1$ をサイクル t0 の状態の出力値とすることで実現する.





図A・5 BUS を用いた CPU の回路図

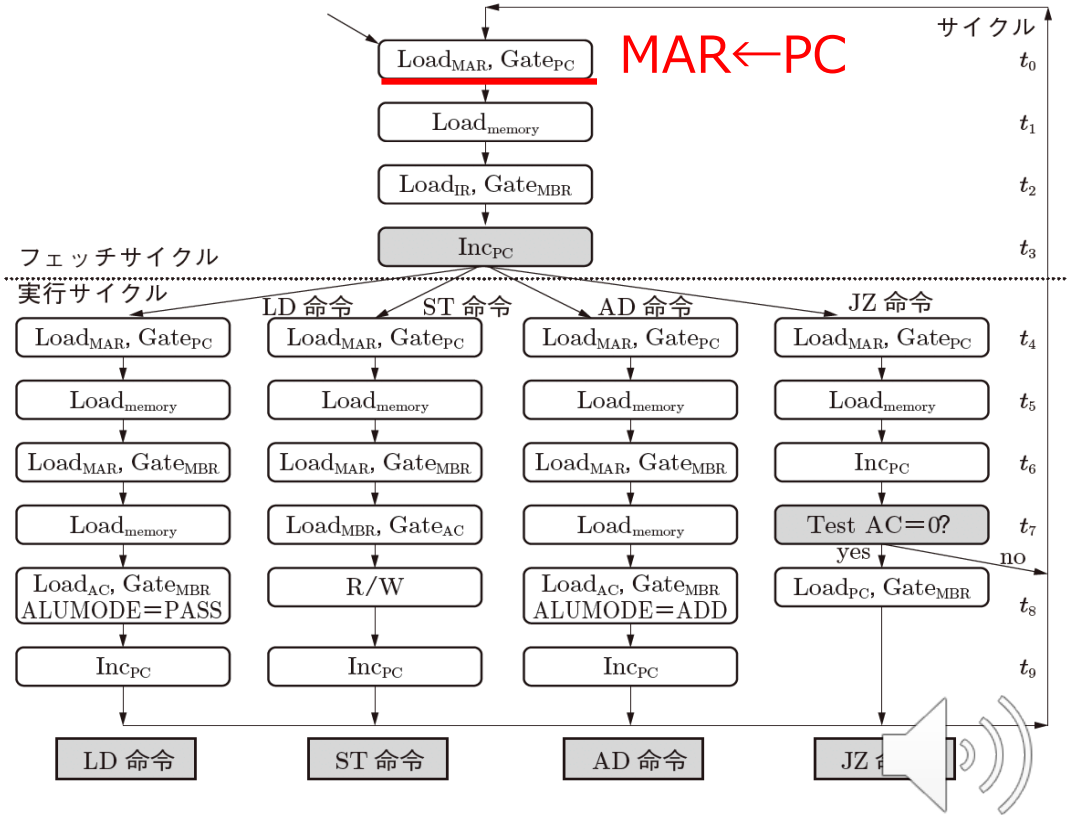
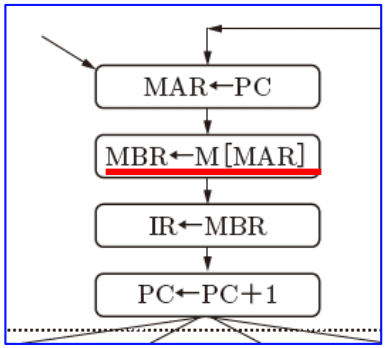
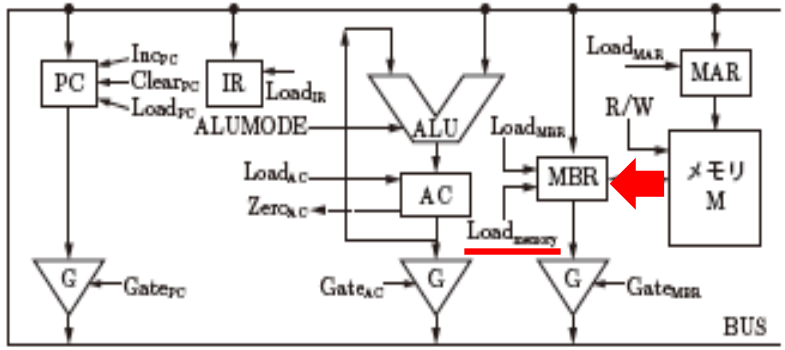


図 A・7 制御部の状態遷移図の出力値

CPU の実現法
～Moore 型順序機械としての実現法～

- フェッチサイクル，実行サイクルの操作列の各操作を実行するためには，基本的に 図A·7 の各状態でこの図の出力値を出力するようにMoore 型順序機械を実現すればよい.
- また，フェッチサイクルの t_1 は $Load_{memory} = 1$ をサイクル t1 の状態の出力値とすることで実現する.





図A・5 BUS を用いた CPU の回路図

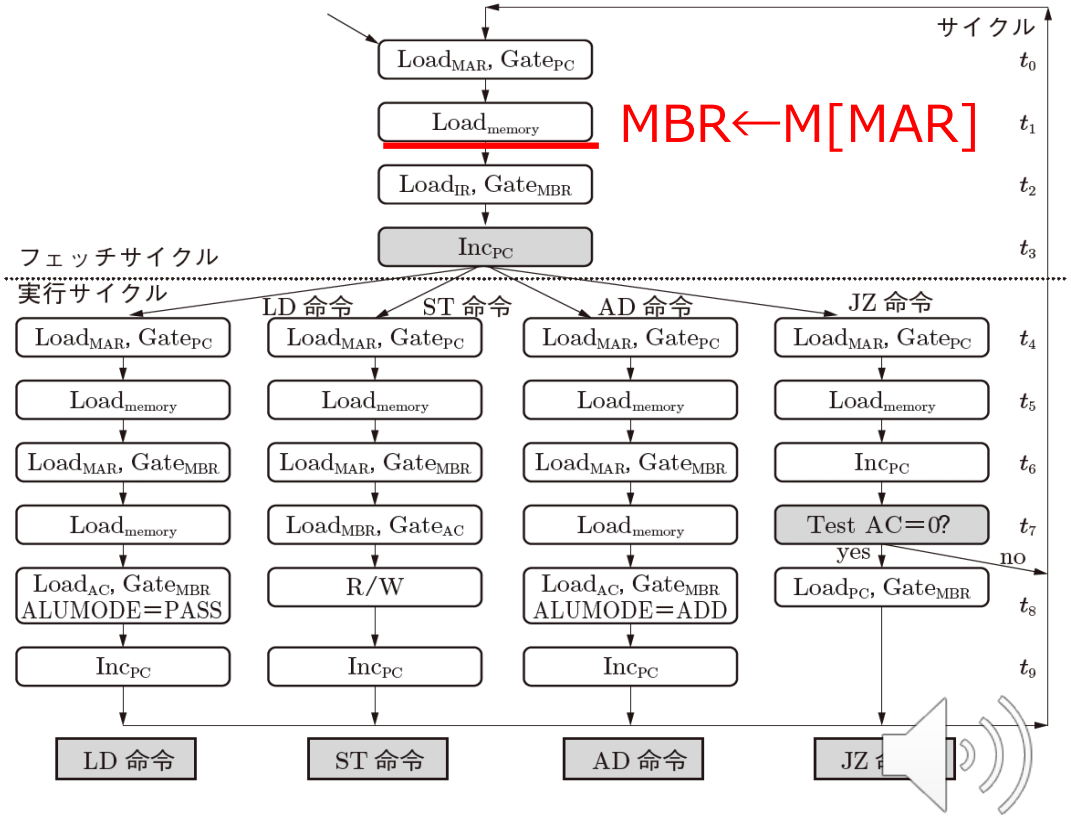


図 A・7 制御部の状態遷移図の出力値

CPU の実現法
～Moore 型順序機械としての実現法～

- 加算 AC←AC+MBR は ALUMODE を ADD に，$Gate_{MBR}$ 信号をオン（$Gate_{MBR}$ = 1）にし，$Load_{AC}$ 信号をオン（$Load_{AC}$ = 1）にすることで実現可能であり，AD 命令のサイクル t_8 の状態では，ALUMODE = ADD, $Gate_{MBR}$ = 1, $Load_{AC}$ = 1 をその出力値とすることでこれらを実現する.

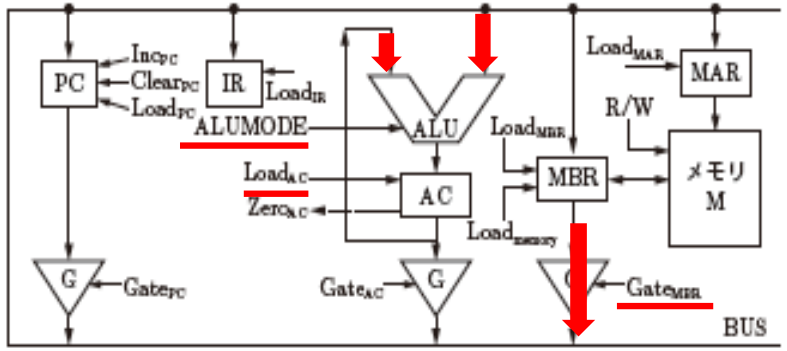


図 A・5 BUS を用いた CPU の回路図

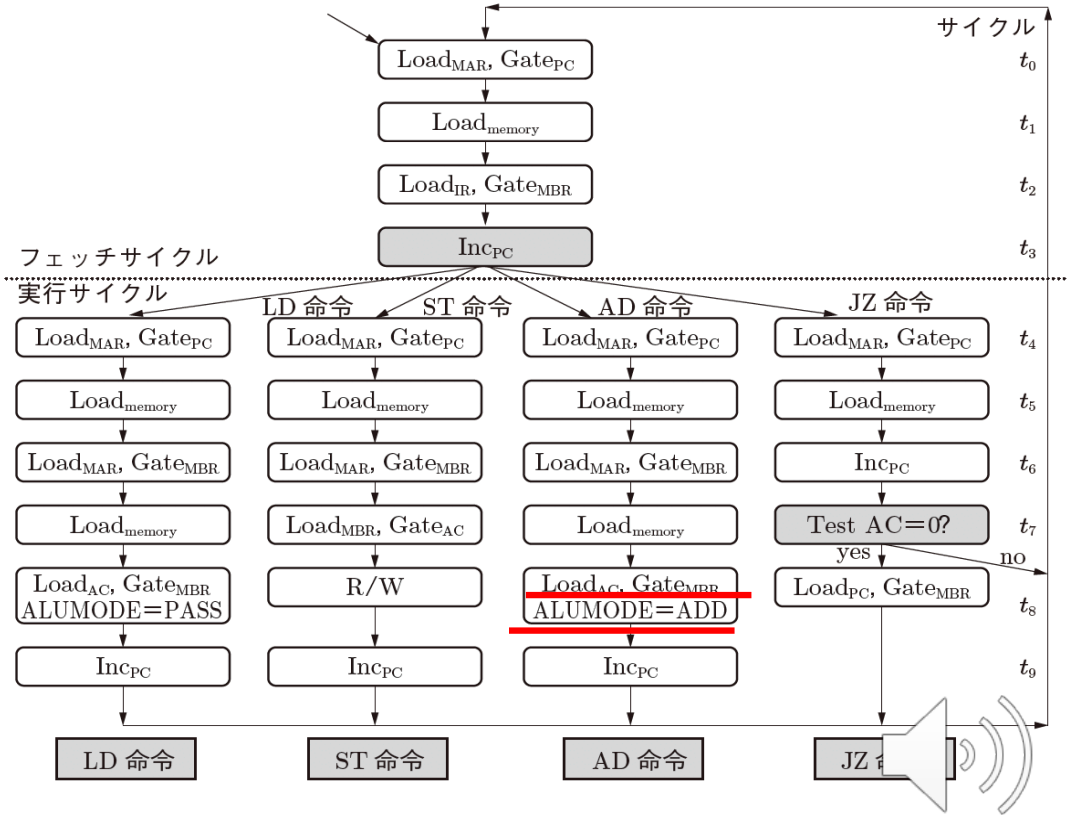


図 A・7 制御部の状態遷移図の出力値

CPU の実現法
～分岐の実現法～

- CPU の順序機械では入力はクロック入力のみであり，図A·7 で灰色で図示されている二つの状態（フェッ チサイクル t_3 の状態，および，JZ 命令のサイクル t_7 の “Test AC=0” を判定する状態）については，命令レジスタ IR の内容に依存して LD, ST, AD, JZ のいずれかの命令に 4 分岐 させたり，“Test AC=0” を判定する部分でレジスタ AC の内容 に依存して 2 分岐させたりする必要が生じる.
- 図A·8（b）に示すように，命令レジスタIRとレジスタ AC が 0 であることを表す $Zero_{AC}$ 信号の内容を用いて，次の状態や制御信号の出力を定める組合せ論理回路を実現する.

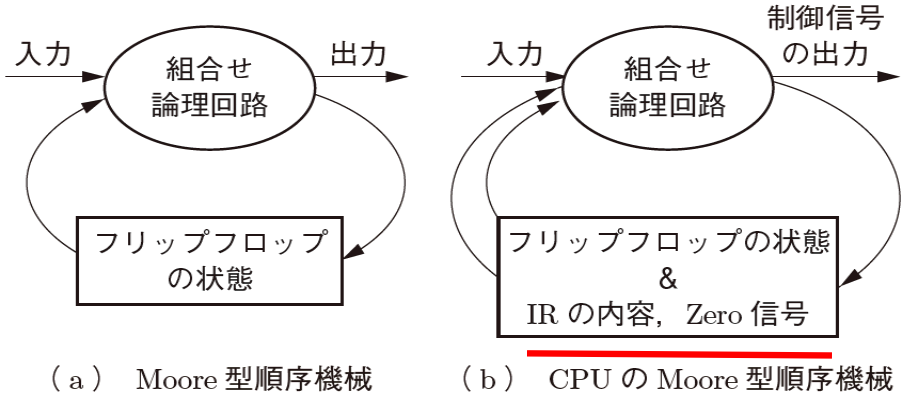


図 A・8 Moore 型順序機械としての実現

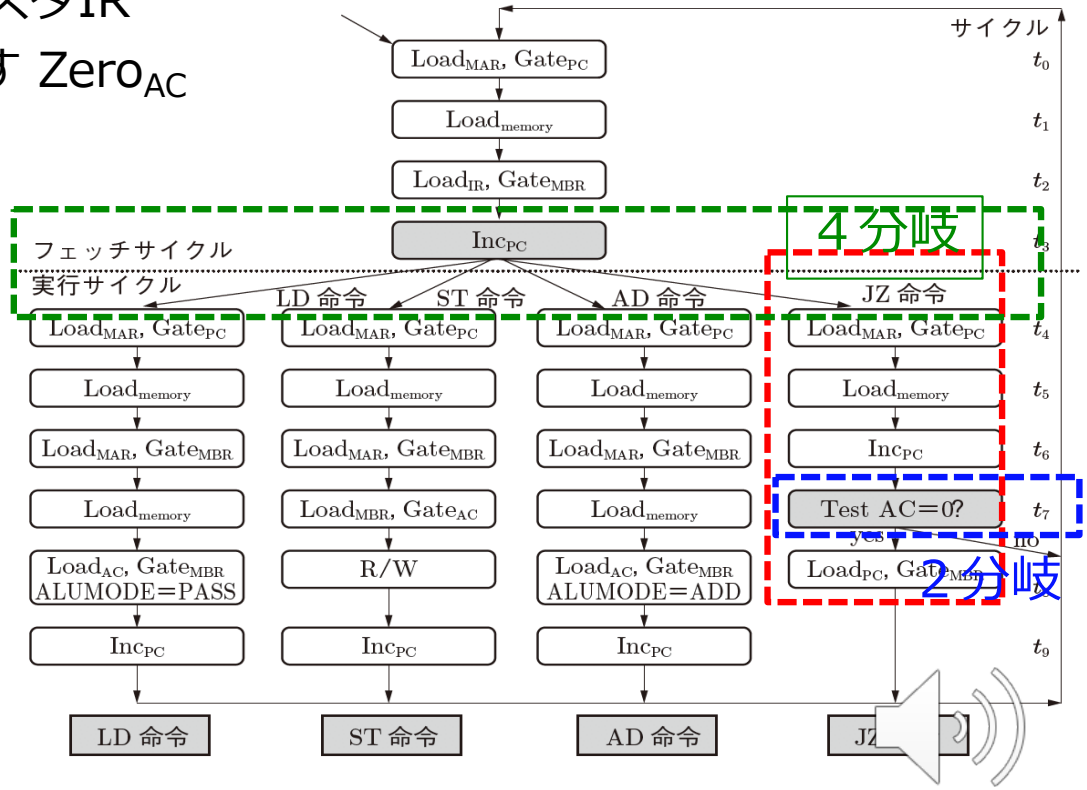
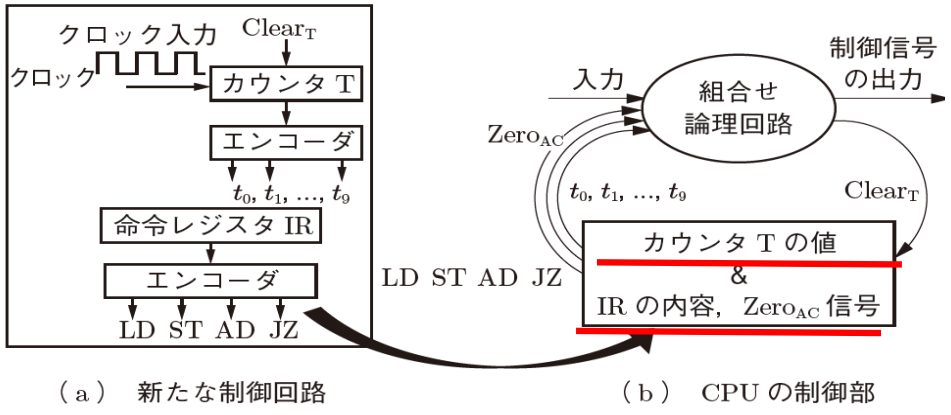


図 A・7 制御部の状態遷移図の出力値

CPU の実現法
～CPUの状態と制御信号の出力の求め方～

- 図A·7の状態遷移図を制御するための方法として，図A·9（a）に示すように，ここでは新たにカウンタ T を設け，クロックが入力されるごとにカウンタ T の値が 1 ずつ増加し，カウンタ T の値が k のとき，対応する制御信号 t_k の値が 1 になるようなエンコーダ回路が実装されているとする．また，この回路では $clear_T$ を 1 にすることで，カウンタ T の値が 0 になり，対応する制御信号 t_0 の値が 1 になるとする．
- 命令レジスタIR にも エンコーダ回路が設けられ，フェッチサイクルで命令レジスタIRに代入される 命令の内容に従って，それぞれ LD, ST, AD, JZ 信号が 1 になるものとする．



（a） 新たな制御回路　　（b） CPU の制御部

図 A·9 CPU の状態と制御信号の出力の求め方

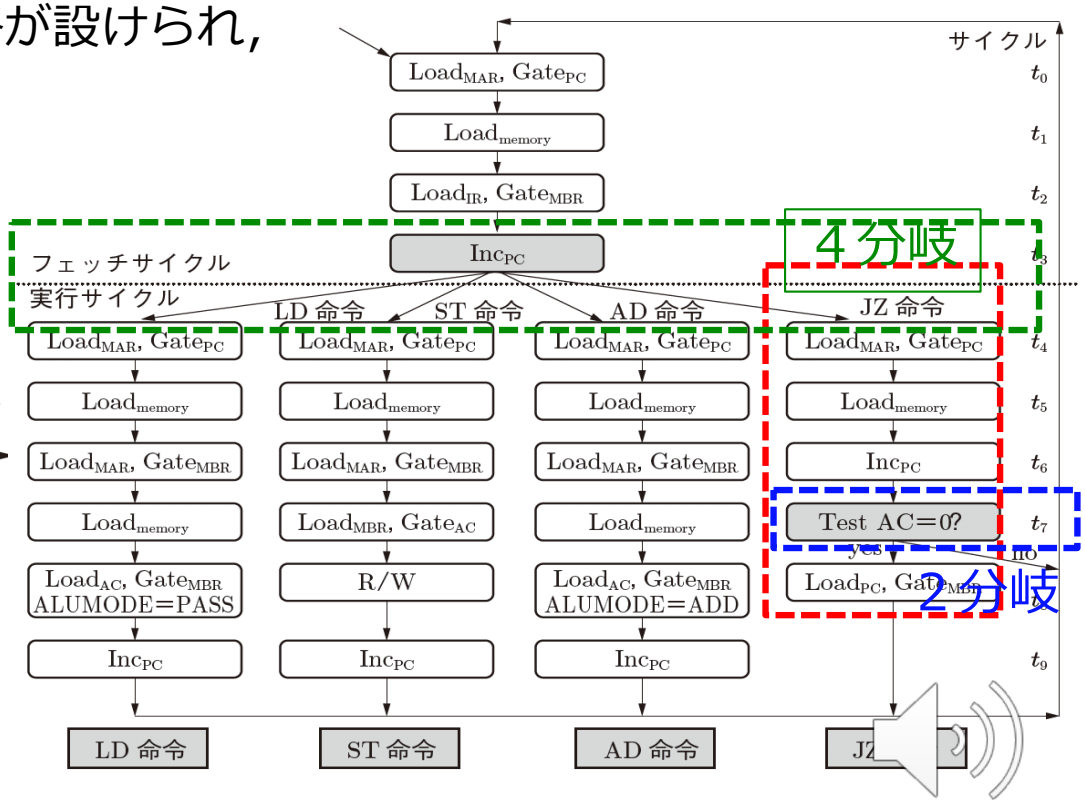


図 A·7 制御部の状態遷移図の出力値

CPU の実現法
～CPUの状態と制御信号の出力の求め方～

- 図A·5 のCPU の回路図の各部品の制御信号の値も図A·7 の各状態の出力値になるよう，制御信号を定めることができる．例えば，$Load_{MAR}$, $Gate_{PC}$, $Load_{momory}$ 信号の値はそれぞれ次のように定めればよい．
  - $Load_{MAR} = t_0 \vee t_4 \vee (t_6 \cdot (LD \vee ST \vee AD))$
  - $Gate_{PC} = t_0 \vee t_4$
  - $Load_{momory} = t_1 \vee t_5 \vee (t_7 \cdot (LD \vee AD))$
- また，$clear_T$ 信号は次のように定義可能である．
  - $clear_T = (t_7 \cdot JZ \cdot \neg Zero_{AC}) \vee t_9$

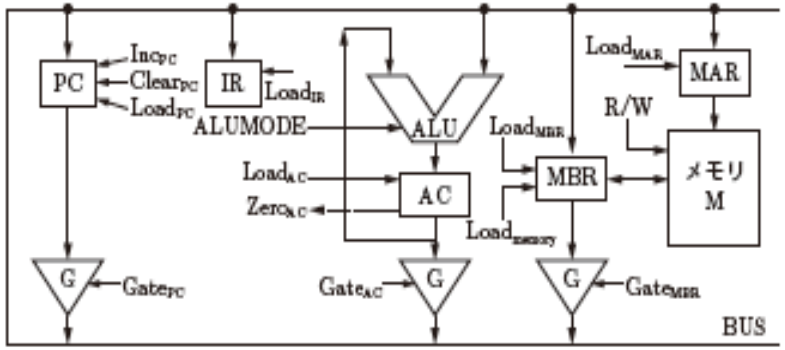


図 A・5　BUS を用いた CPU の回路図

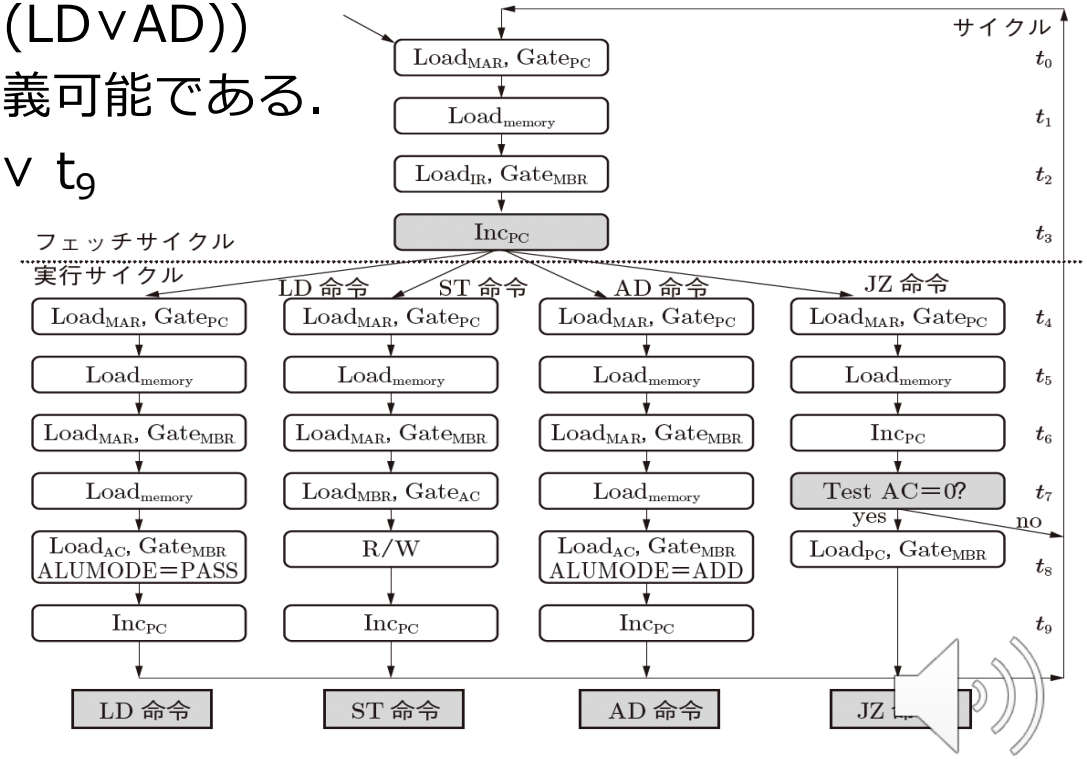


図 A・7　制御部の状態遷移図の出力値

OSAKA UNIVERSITY

12回目の授業終了

授業終了

皆さん
今日はレポート課題はありません