論理設計
東野担当13回目
授業スライド
１月６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（変更）：
  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. １月２０日は授業なし（この日は対面での試験の実施日）
  16. 期末試験（１２章～１５章，付録）（期末試験は１月２７日に実施）

期末試験も中間試験同様
オンラインで実施予定

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.

付録
CPUの設計
（前回の復習）

設計するCPU の概要

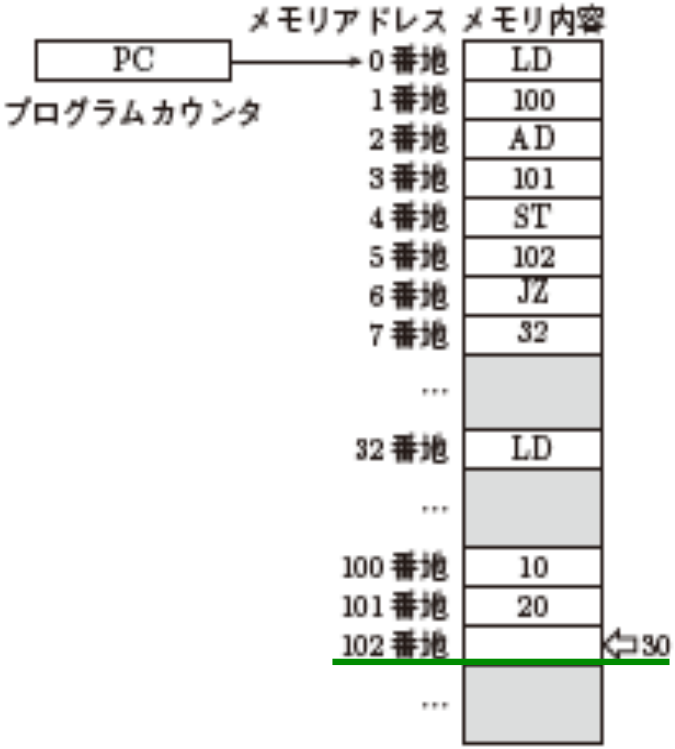
- 機械語命令の指定
  - CPU（Central Processing Unit）はコンピュータの中心的な回路であり，中央処理装置とも呼ばれ，機械語で書かれたプログラムを実行することで，さまざまな数値計算や情報処理を行う．機械語の命令には，ロード（LD），ストア（ST），加算（AD），条件付ジャンプ（JZ）などいくつかの種類がある．
  - 本章では四つの命令（LD, ST, AD, JZ）からなるCPU の設計を考える．以下では，各命令は下記のような2 語命令（各inC_k = 2）とし，図A·1 のようにメモリの最初の 1 語にその命令の名前が書かれ，次の 1 語にその命令のオペランド（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

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）

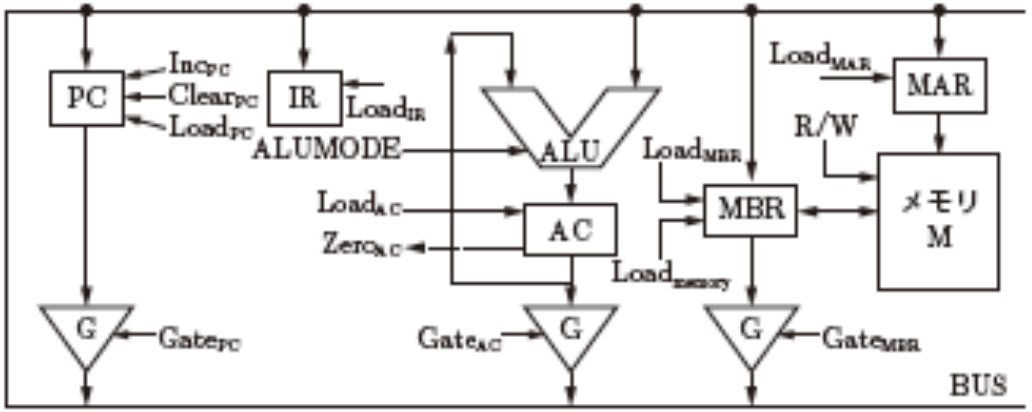
- 図A·1 にメモリに格納された機械語プログラムの例を示す．この例では，最初プログラムカウンタ PC は 0 番地を指しているものとする．0 番地の命令がロード（LD）命令で 1 番地に 100 が書かれているので，この 2 語命令を実行すると，100 番地の内容 10 がアキュムレータ AC に格納され，プログラムカウンタ PC の値が 2 増加する．
- PC が 2 番地の AD 命令を指し 3 番地に 101 が書かれているので，AD 命令が実行され，アキュムレータ AC に 101 番地の内容 20 が加算される（30 に変化）．
- その後プログラムカウンタ PC の値が 2 増加して 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）

CPU 回路の構成

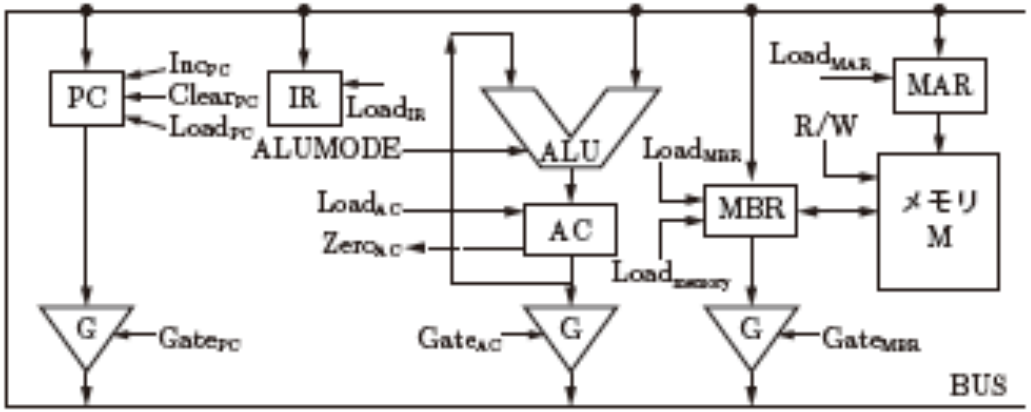
- 以下では CPU回路を 図A·5 のように，メモリ (M)，プログラムカウンタ(PC)，算術論理演算装置(ALU)，レジスタ(MBR, MAR, AC)，命令レジスタ(IR)，三つのゲート回路(G)を一つのバス(BUS)でつないで実現する.
- 各レジスタ間の転送は送信側レジスタ(PC, AC, MBR)のGate 信号($Gate_{PC}$, $Gate_{AC}$, $Gate_{MBR}$) のいずれか一つの信号をオンにし，受信側レジスタ(PC, IR, AC, MBR, MAR)のLoad 信号($Load_{PC}$, $Load_{IR}$, $Load_{AC}$, $Load_{MBR}$, $Load_{MAR}$)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.



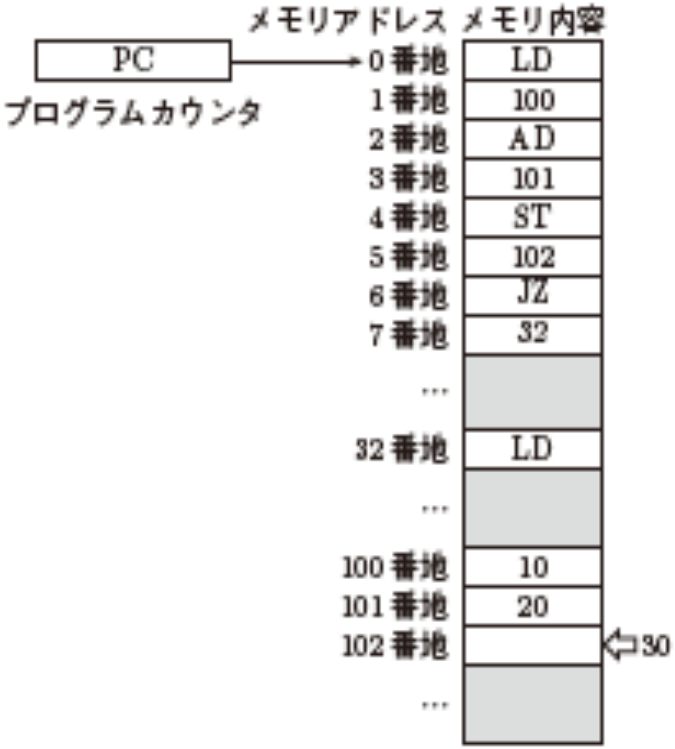
図A・5 BUS を用いた CPU の回路図

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

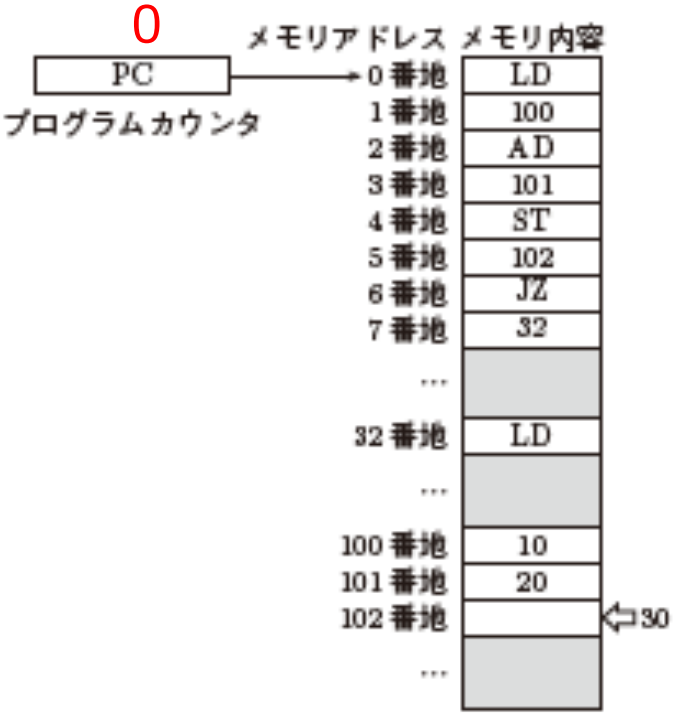
- メモリーの偶数番地に格納されている命令を命令レジスタ IR に転送する.
- フェッチサイクルでは，次の (t_0) から (t_3) の四つの操作を順に実行することで，命令レジスタ IR に命令の名前が格納され，プログラムカウンタ PC がその命令のオペランドを読み出せるようになる.

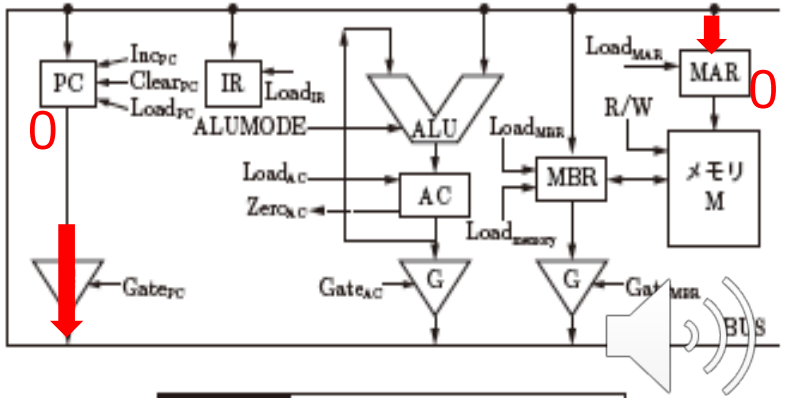
(t_0) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送（MAR←PC）

(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

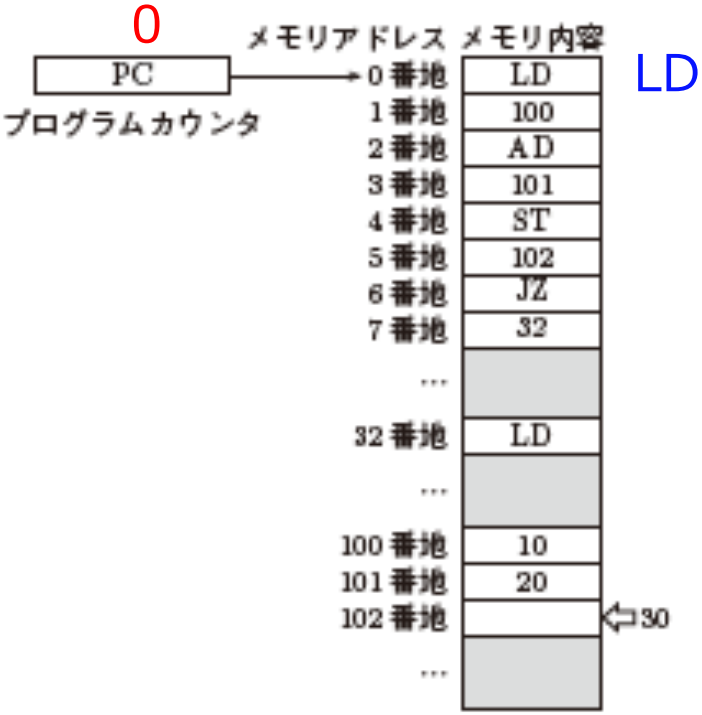
- メモリーの偶数番地に格納されている命令を命令レジスタ IR に転送する.
- フェッチサイクルでは，次の (t_0) から (t_3) の四つの操作を順に実行することで，命令レジスタ IR に命令の名前が格納され，プログラムカウンタ PC がその命令のオペランドを読み出せるようになる.

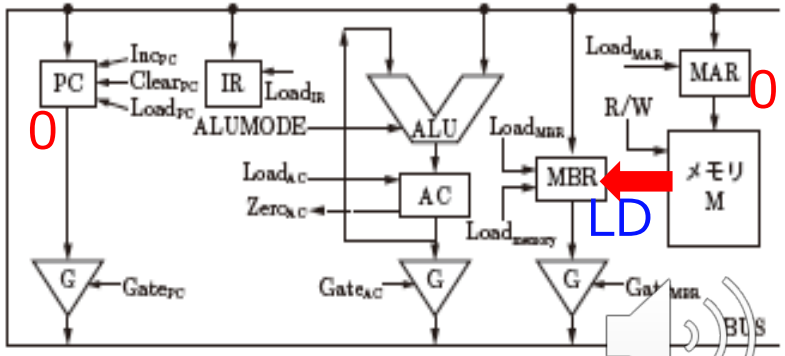
(t_0) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送（MAR←PC） 0

(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

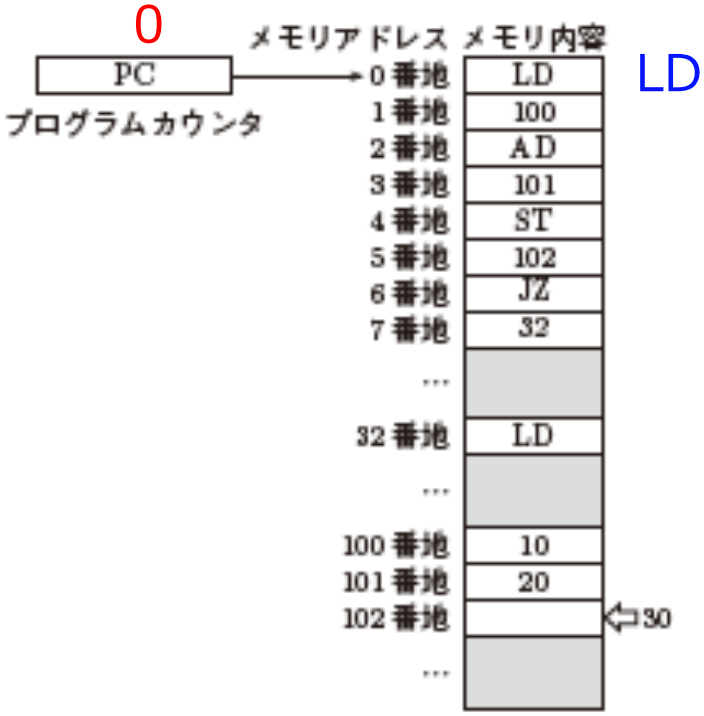
- メモリーの偶数番地に格納されている命令を命令レジスタ IR に転送する.
- フェッチサイクルでは，次の (t_0) から (t_3) の四つの操作を順に実行することで，命令レジスタ IR に命令の名前が格納され，プログラムカウンタ PC がその命令のオペランドを読み出せるようになる.

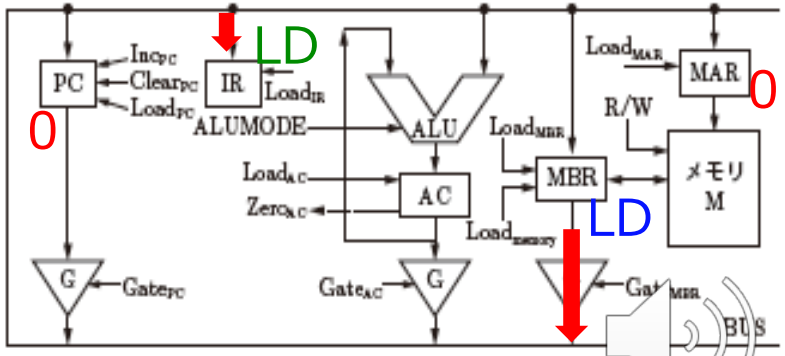
(t_0) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送（MAR←PC） 0

(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

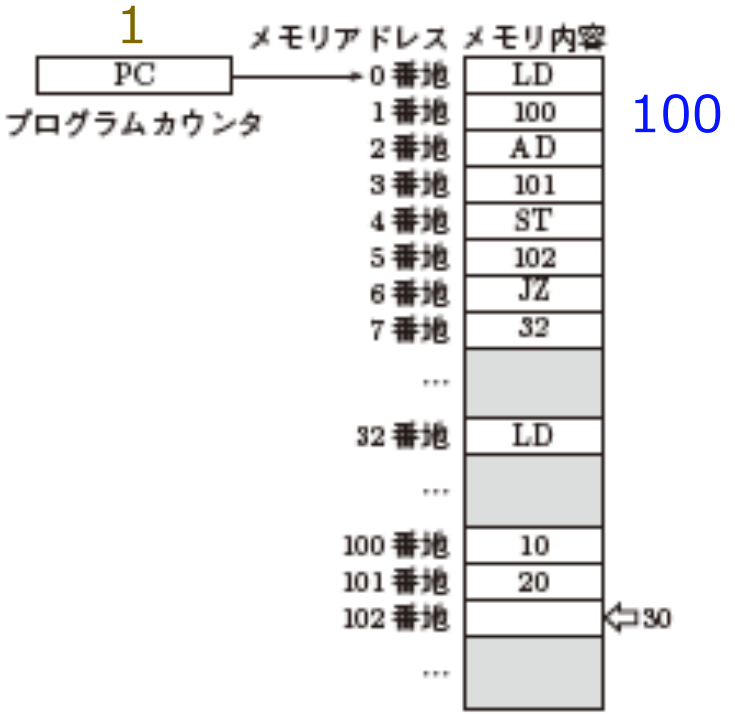
- メモリーの偶数番地に格納されている命令を命令レジスタ IR に転送する.
- フェッチサイクルでは，次の (t_0) から (t_3) の四つの操作を順に実行することで，命令レジスタ IR に命令の名前が格納され，プログラムカウンタ PC がその命令のオペランドを読み出せるようになる.

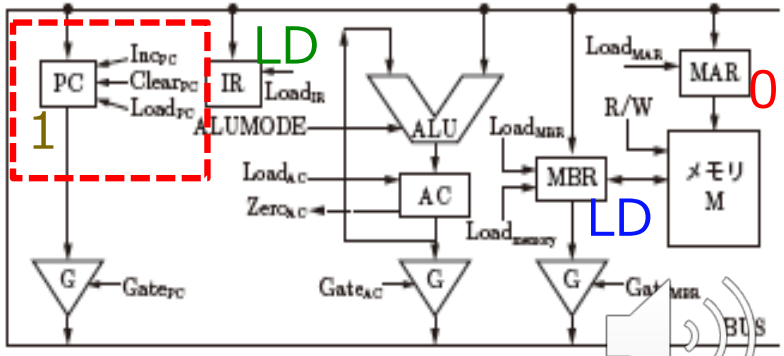
(t_0) まずプログラムカウンタ PC の値をメモリ M のアドレスを指定するレジスタ MAR に転送（MAR←PC） 0

(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

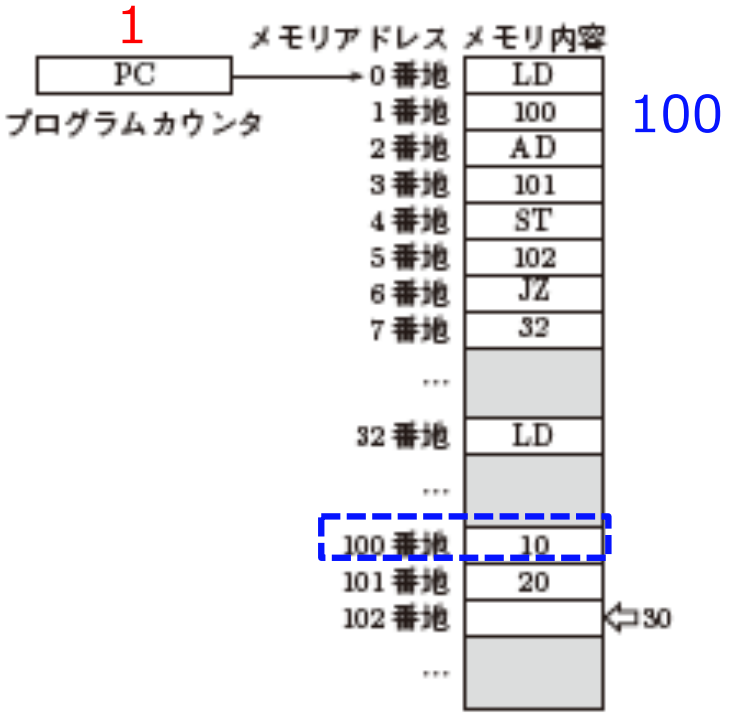
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの adr 番地を取得

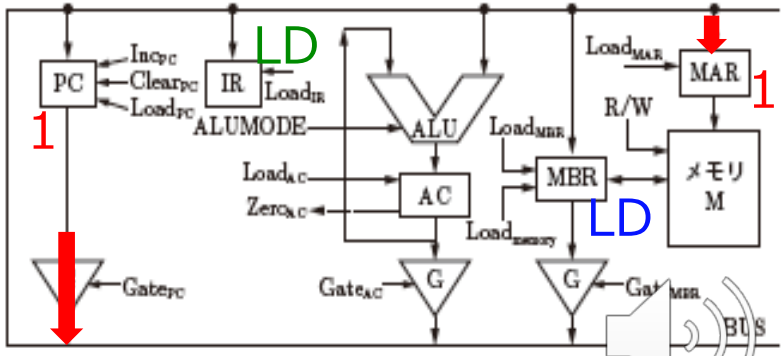
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR）

(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）

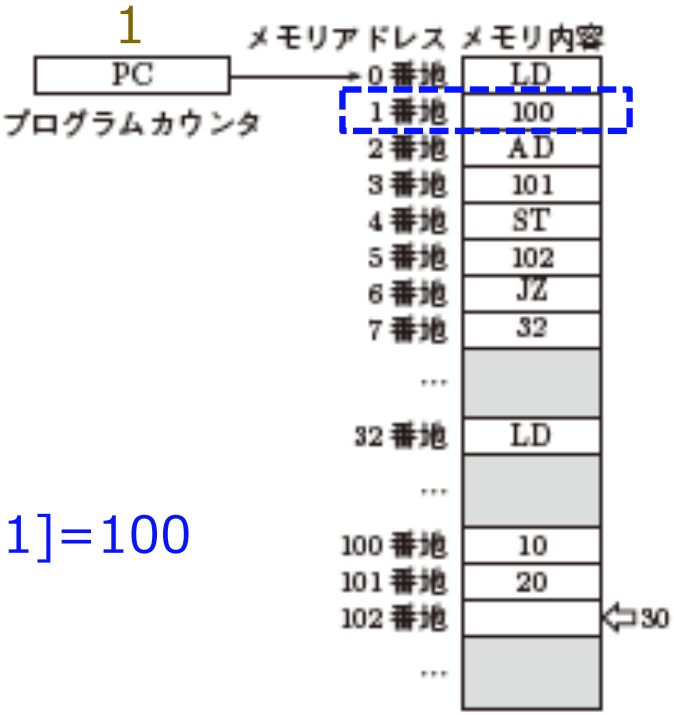
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）オペランドの adr 番地を取得　M[1]=100

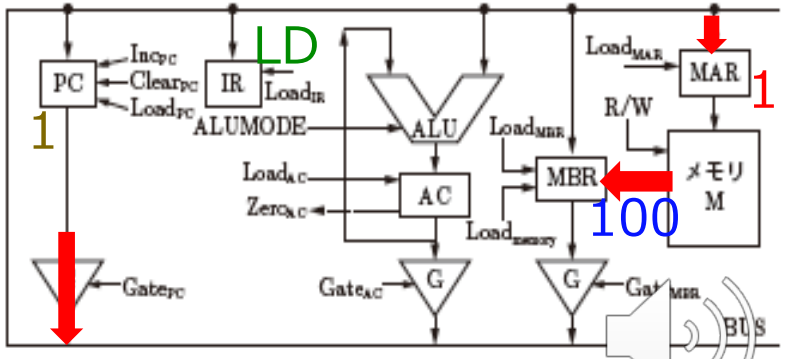
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR）

(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）

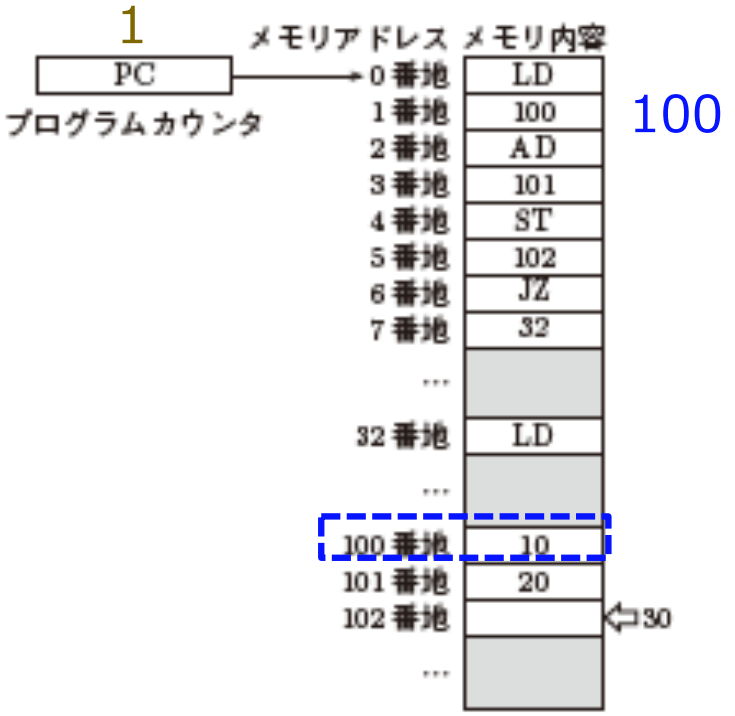
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの adr 番地を取得

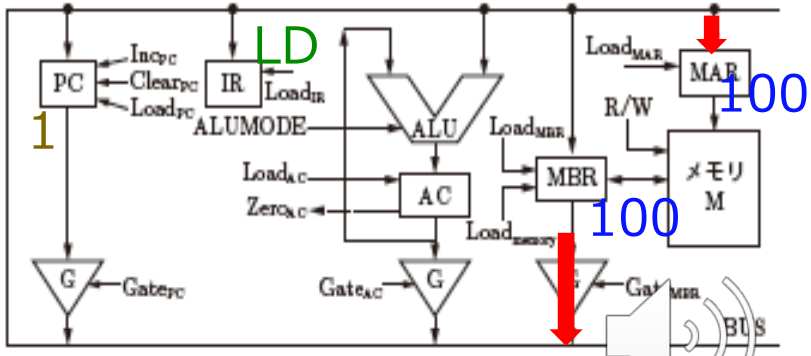
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR） 100

(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）

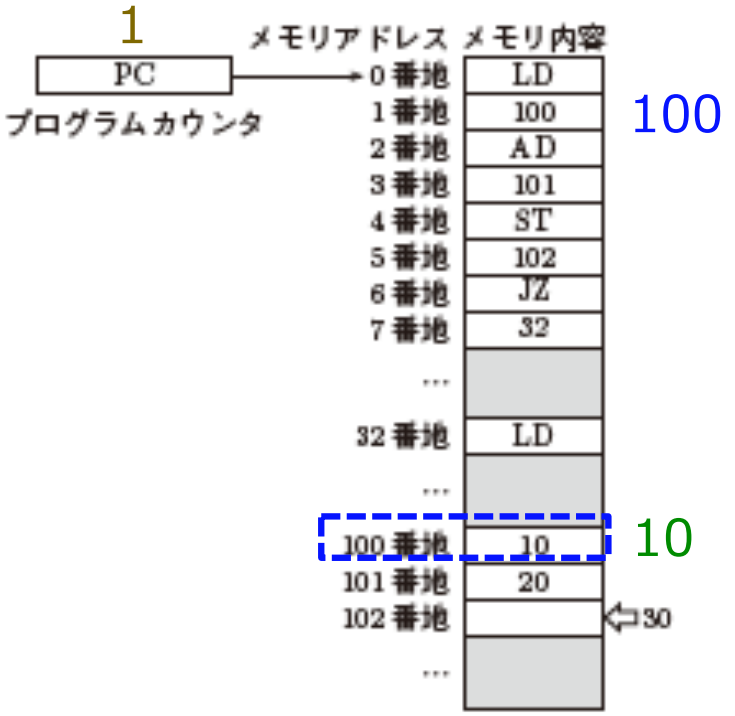
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの adr 番地を取得

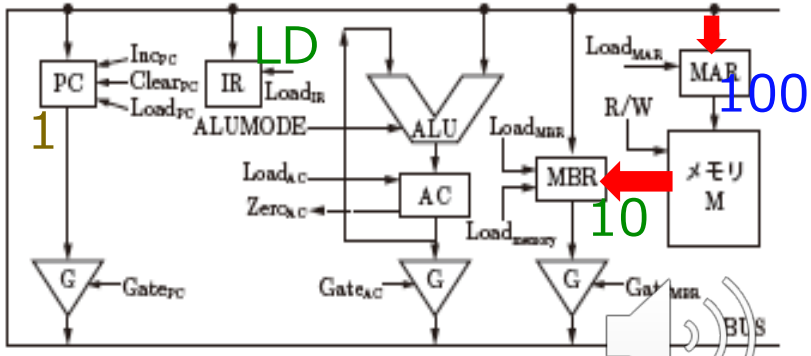
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR） 100

(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）

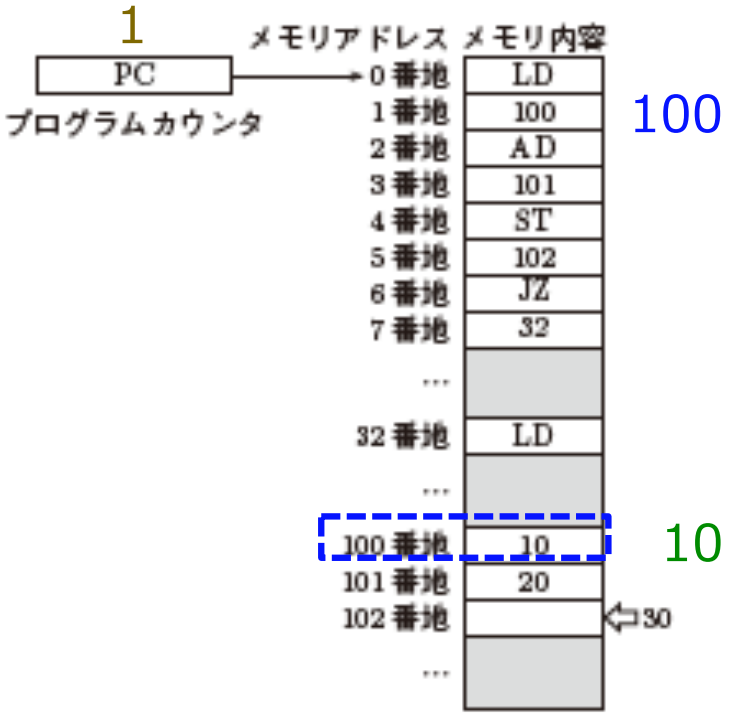
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの adr 番地を取得

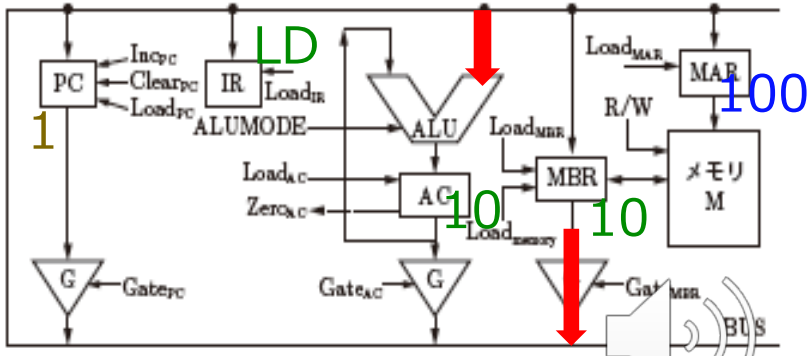
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR） 100

(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）

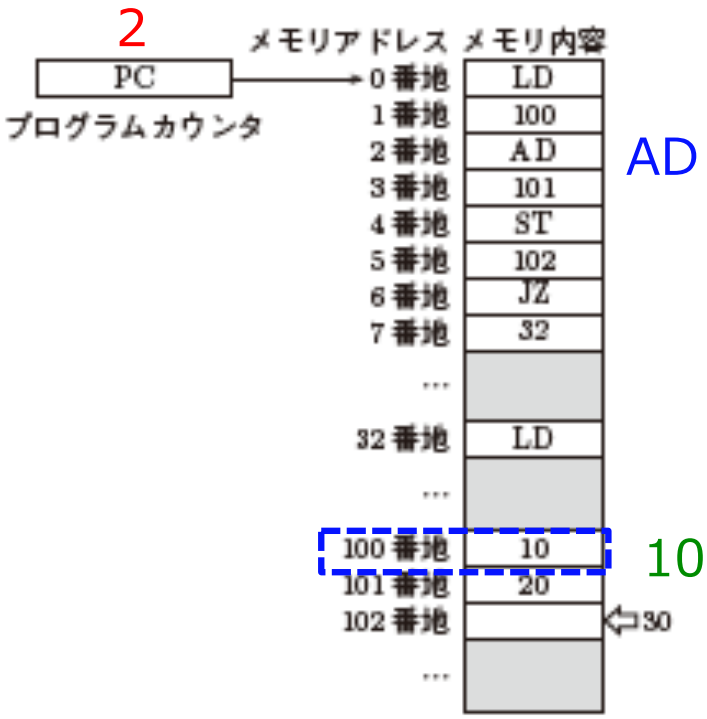
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの adr 番地を取得

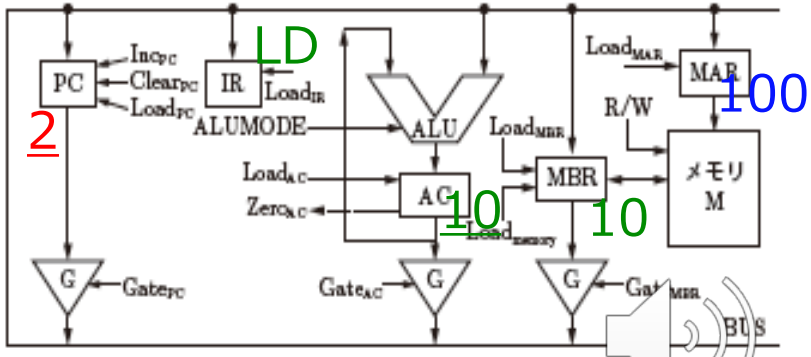
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR） 100

(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）

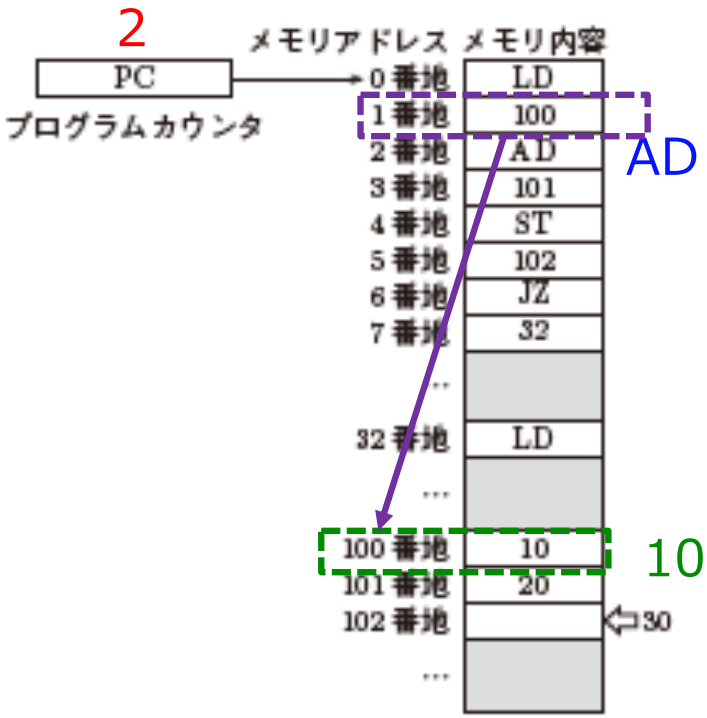
(t_5) MAR番地の内容 M[MAR] をレジスタ MBR に転送
（MBR←M[MAR]）：オペランドの adr 番地を取得

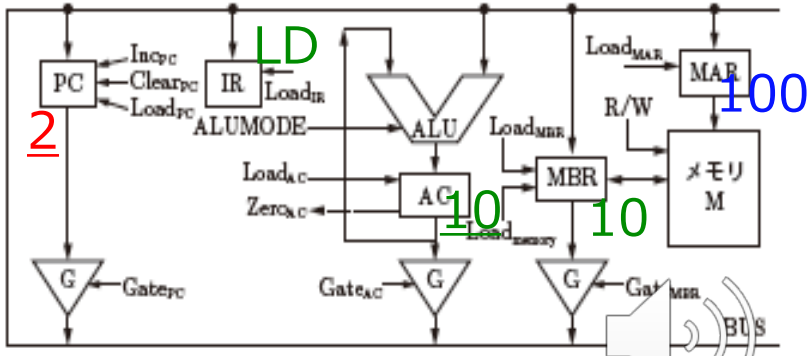
(t_6) レジスタ MBR の内容（オペランドに記載の adr 番地）をメモリアドレスレジスタ MAR に転送
（MAR←MBR） 100

(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

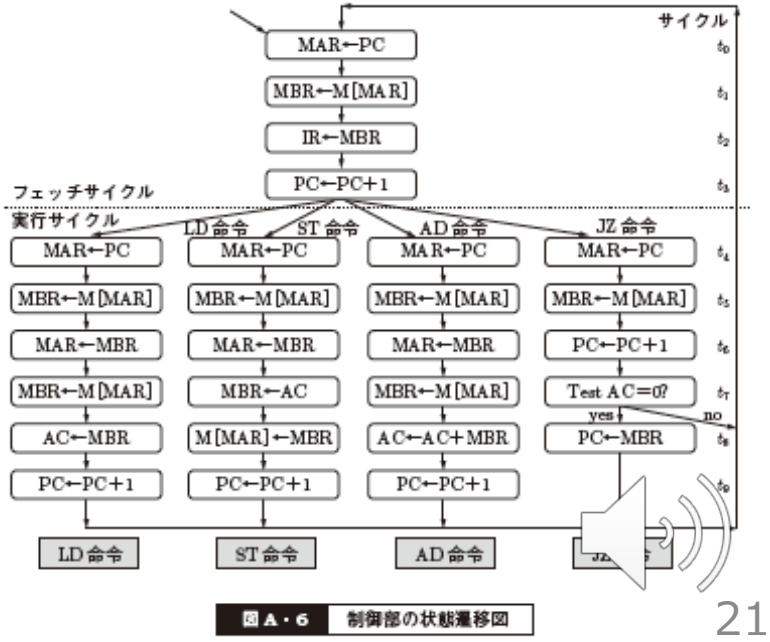




図A・5 BUS を用いた CPU の回路図

CPU の実現法
～Moore 型順序機械としての実現法～

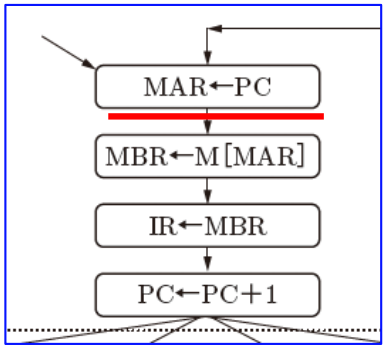
- 上述のフェッチサイクルや実行サイクルの実行順は図A·6 の状態遷移図に記載の Moore 型順序機械で実現できる．図A·6 では，フェッチサイクルが 4 サイクル（t_0,…, t_3），実行サイクルが 6 サイクル（t_4,…,t_9）または 5 サイクル（t_4,…,t_8）で実現され，フェッチサイクルと実行サイクルを順に繰り返し実行することで，機械語のプログラムを順次解釈・実行していく（Moore 型順序機械の各状態の出力については後述する）．
- 上記のフェッチサイクルや実行サイクルは，レジスタ間のデータ転送を行ったり，ALU を用いて加算を行ったり，PC の値を 1 増加する，と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.



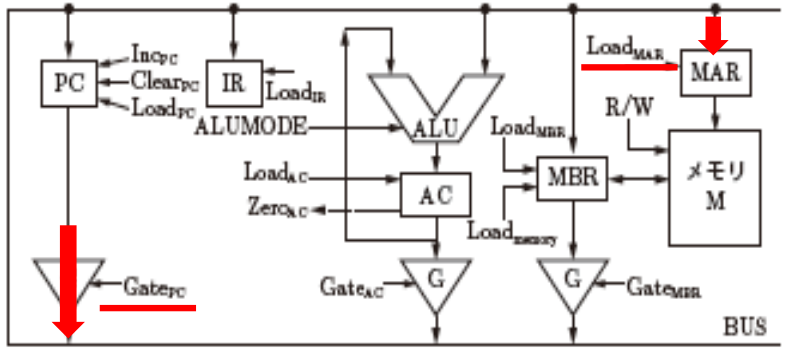


図 A・5 BUS を用いた CPU の回路図

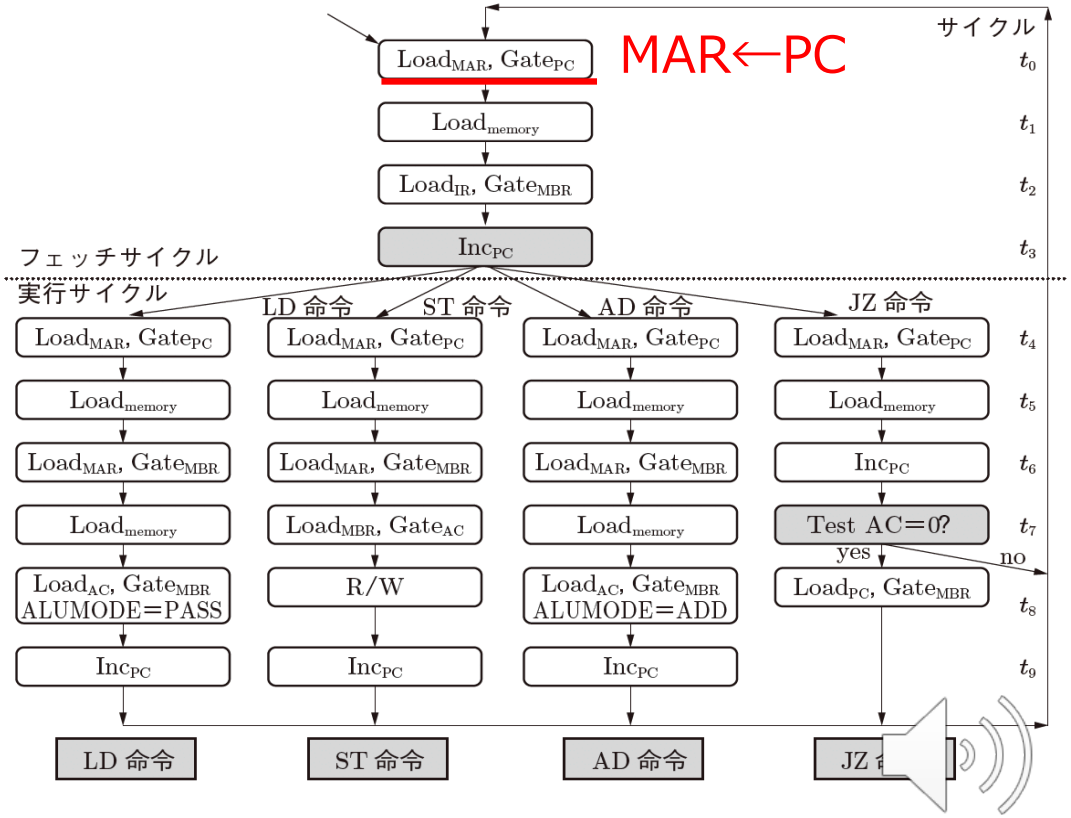
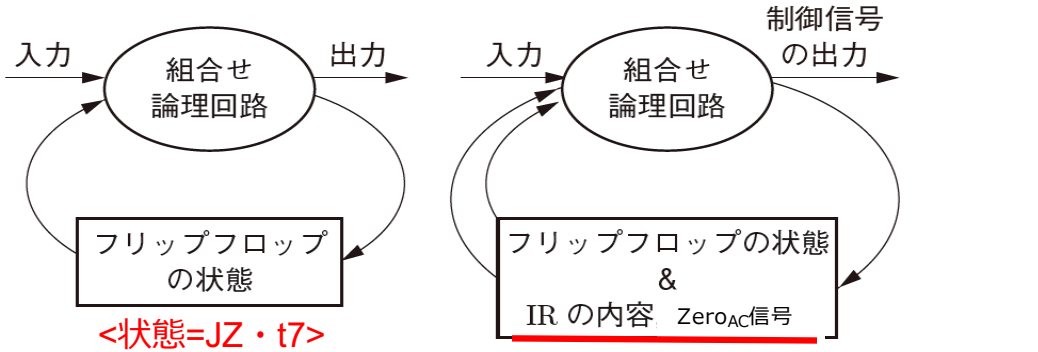


図 A・7 制御部の状態遷移図の出力値

CPU の実現法
～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.



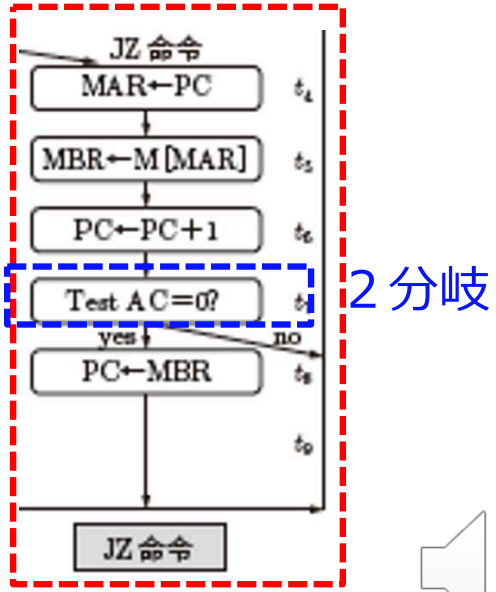
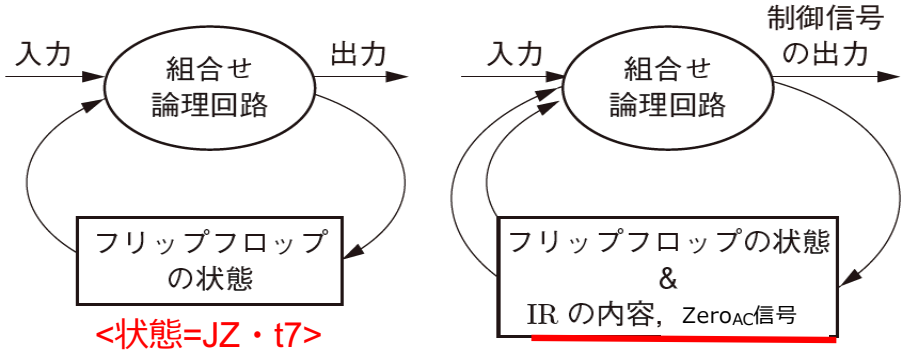


図 A・8 Moore 型順序機械としての実現

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.



（a） Moore 型順序機械　（b） CPU の Moore 型順序機械

<<状態=JZ・t7>,<IR=JZ>,<$Zero_{AC}$=0>>

図 A・8 Moore 型順序機械としての実現

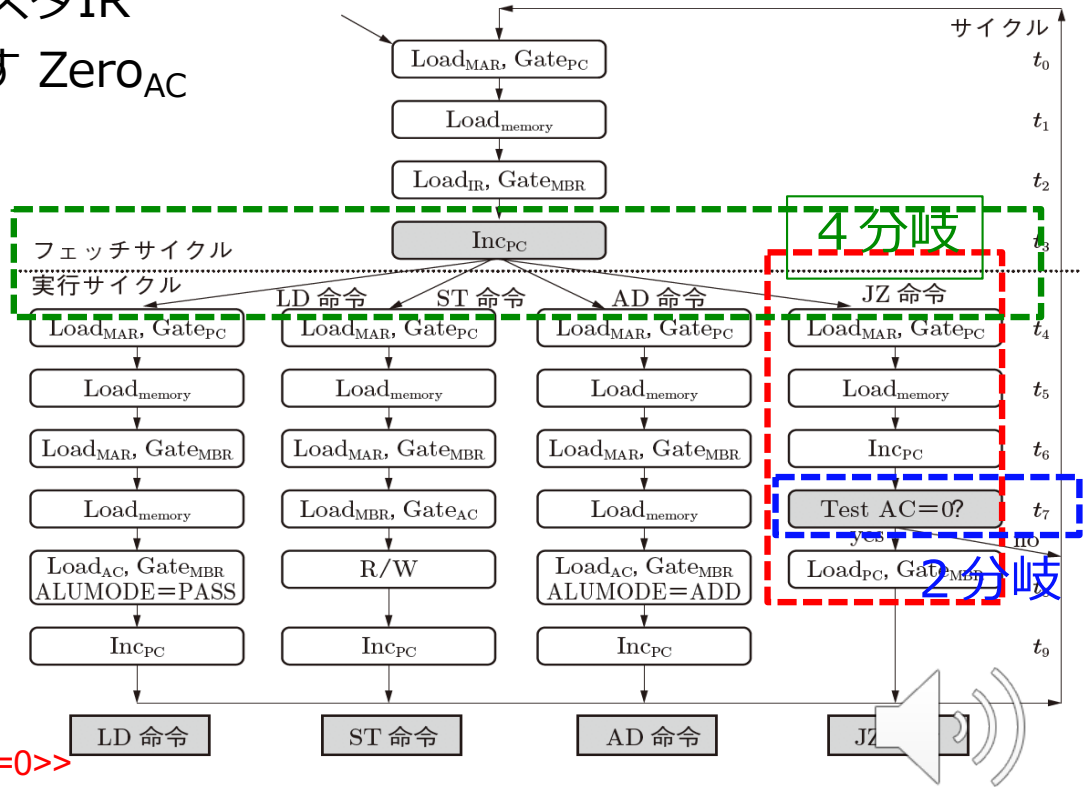


図 A・7 制御部の状態遷移図の出力値

CPU の実現法
～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.

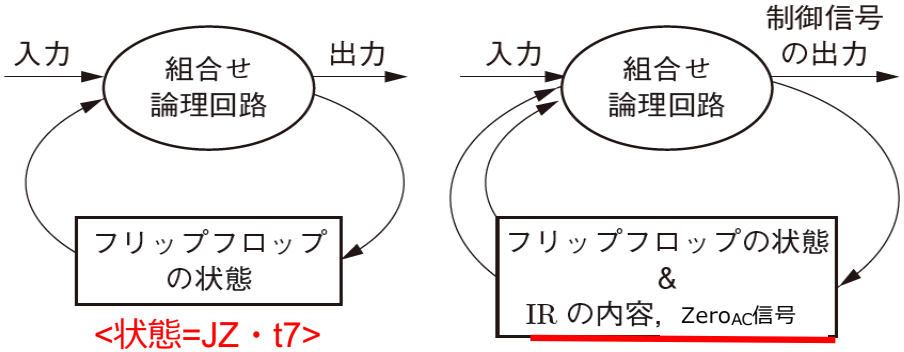


図 A・8 Moore 型順序機械としての実現

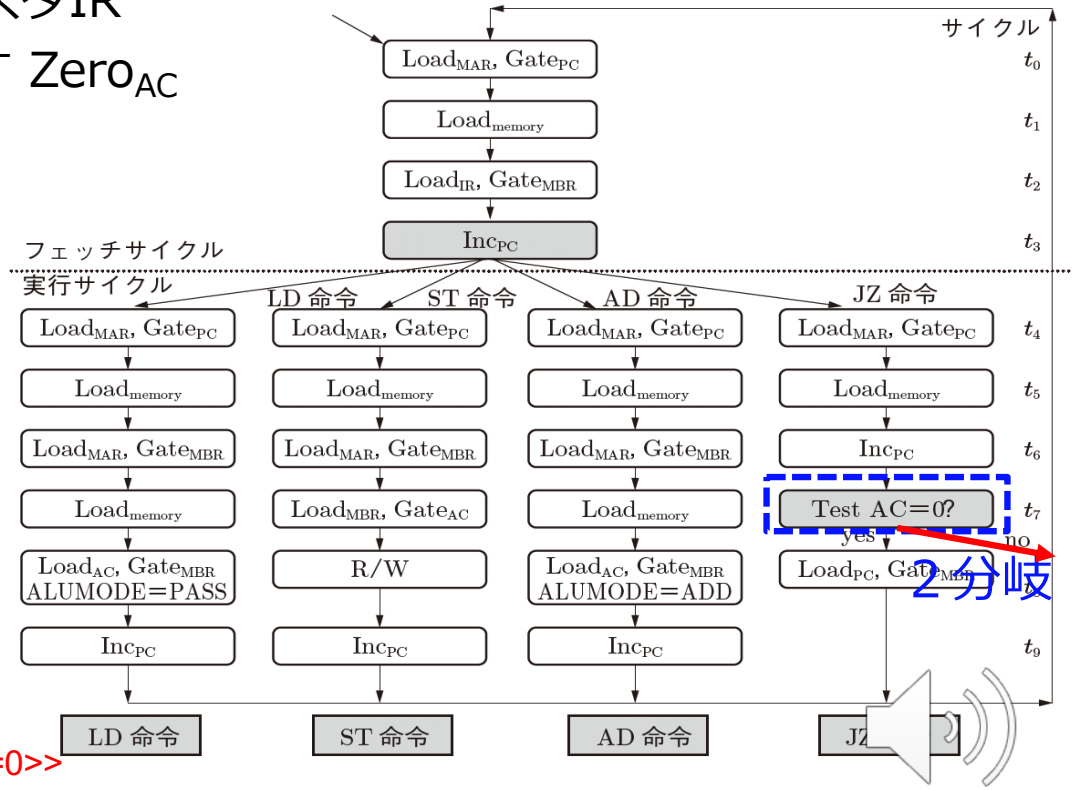
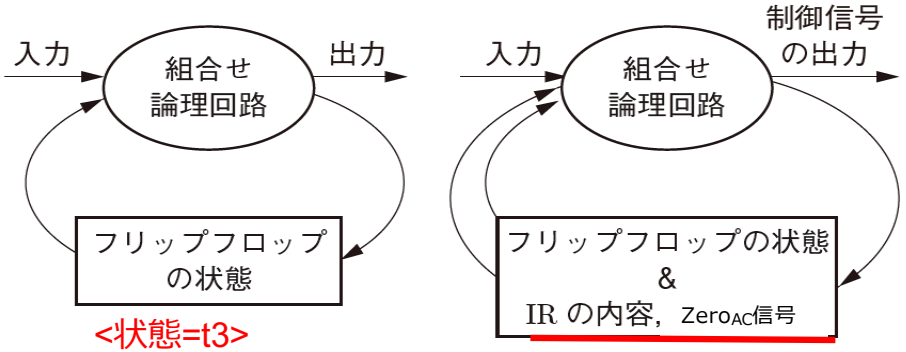


図 A・7 制御部の状態遷移図の出力値

CPU の実現法
～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.



（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

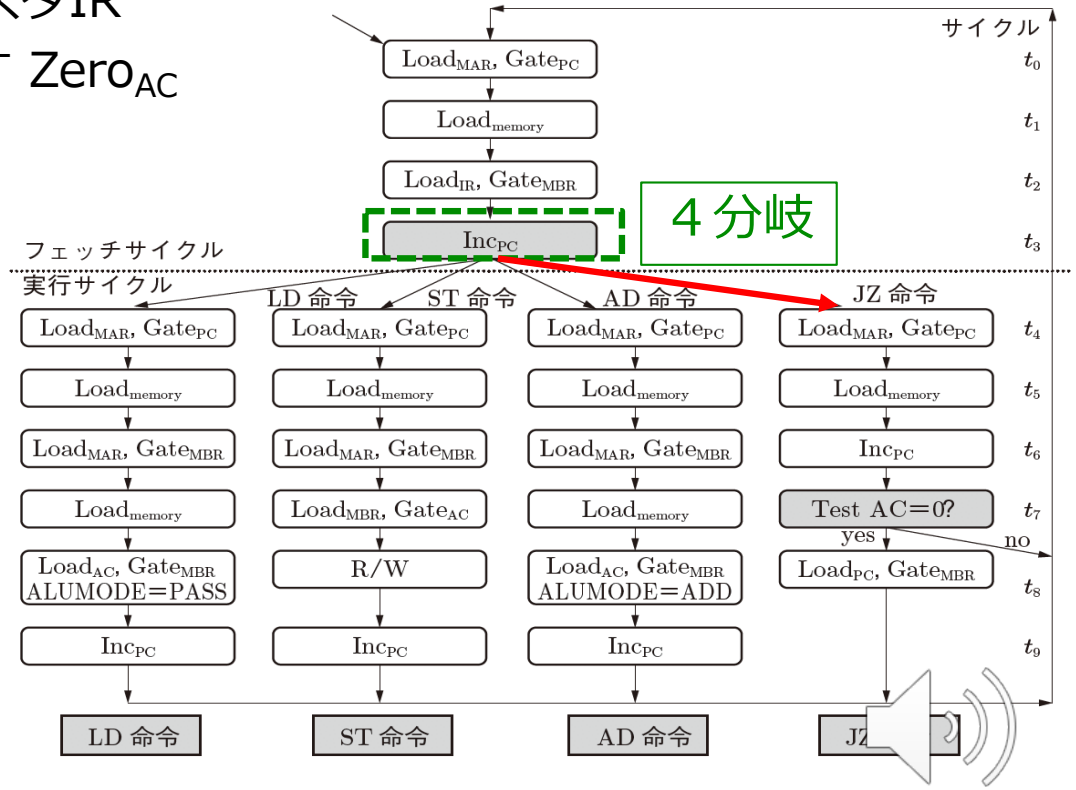
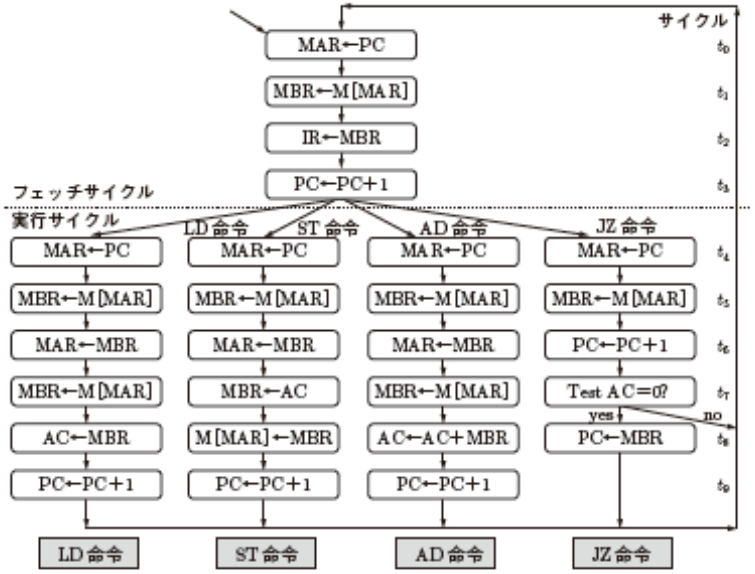


図 A・7 制御部の状態遷移図の出力値

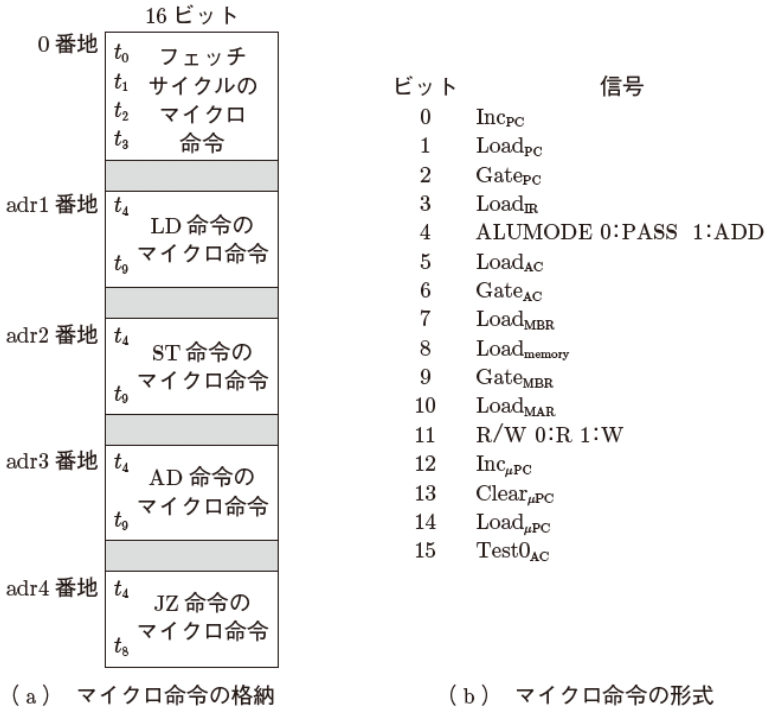
付録
CPUの設計
（219頁以降）

CPU の実現法
～マイクロプログラム方式での実現法～

- 以下では，図A·6 のMoore 型順序機械の状態遷移や制御信号の出力を 15 章で説明したマイクロプログラム方式で実現する方法を説明する．
- 図A·10（a）はそのアドレス指定である．ここでは，フェッチサイクルのマイクロ命令は 0 番地から実行されると考え，“Test AC=0 ?” は，$Zero_{AC}$ 信号がオン（$Zero_{AC}$ = 1）のときはなにも実行せず，オフ（$Zero_{AC}$ = 0）のときはマイクロプログラムのアドレスを 0 番地にクリアすることにより実現する．



図A・6 制御部の状態遷移図

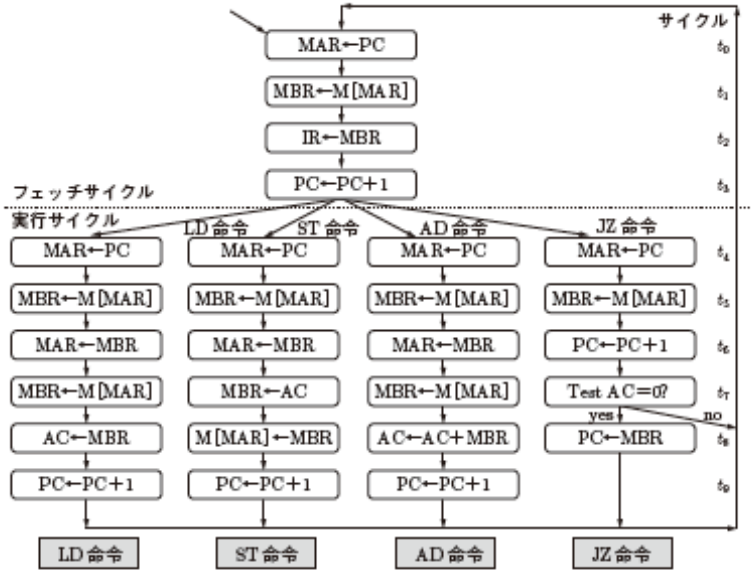


（a） マイクロ命令の格納　　（b） マイクロ命令の形式

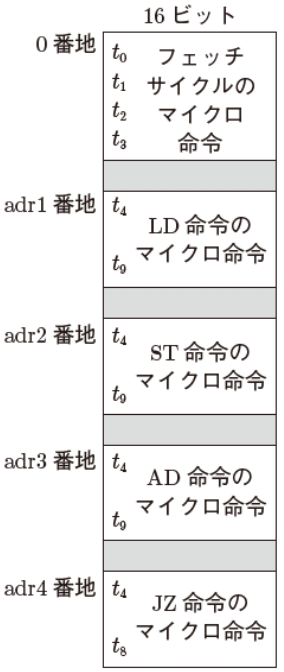
図 A・10 マイクロ命令の格納と形式

CPU の実現法
～マイクロプログラム方式での実現法～

- 以下では，図A·6 のMoore 型順序機械の状態遷移や制御信号の出力を 15 章で説明したマイクロプログラム方式で実現する方法を説明する．
- $Gate_{PC}$ 信号や $Load_{MBR}$ 信号のような信号のオン・オフは，マイクロプログラムによって実現する．マイクロ命令の形式は 図A·10（b）のような 16 ビットのものとする．ビット 12～15 は，フェッチサイクルの t_3 から実行サイクルの t_4 に 遷移する際の 4 分岐や条件付きジャンプ命令 JZ の 2 分岐を制御するために用いる．これらの用法については以下で述べる．



図A・6 制御部の状態遷移図



（a） マイクロ命令の格納

ビット	信号
0	Inc_{PC}
1	$Load_{PC}$
2	$Gate_{PC}$
3	$Load_{IR}$
4	ALUMODE 0:PASS 1:ADD
5	$Load_{AC}$
6	$Gate_{AC}$
7	$Load_{MBR}$
8	$Load_{memory}$
9	$Gate_{MBR}$
10	$Load_{MAR}$
11	R/W 0:R 1:W
12	$Inc_{\mu PC}$
13	$Clear_{\mu PC}$
14	$Load_{\mu PC}$
15	$Test0_{AC}$

（b） マイクロ命令の形式

図A・10 マイクロ命令の格納と形式

CPU の実現法
～μアドレスの指定～

- マイクロプログラムのアドレス指定を 図A·11 のような回路を用いて行う． 図A·11 において，μP–ROM はマイクロプログラムを格納するための ROM であり，そのアドレスをマイクロプログラムカウンタ μPC で指定する．
- μPC の $Inc_{\mu PC}$ 信号によってμPC の値が 1 増加し，$Clear_{\mu PC}$ 信号によって μPC の値が 0 になる． また，μA–ROMには，各命令の実行サイクルのマイクロ命令の開始番地（図A·10（a）の adr1 番地～adr4 番地）が書き込まれており，μA–ROM からは常に IR の命令に対する開始番地が出力される．この開始番地は，μPC の $Load_{\mu PC}$ 信号によって μPC にロードされる．通常のマイクロ命令では，$Inc_{\mu PC}$ 信号によって μPC の値を 1 ずつ増加させる．

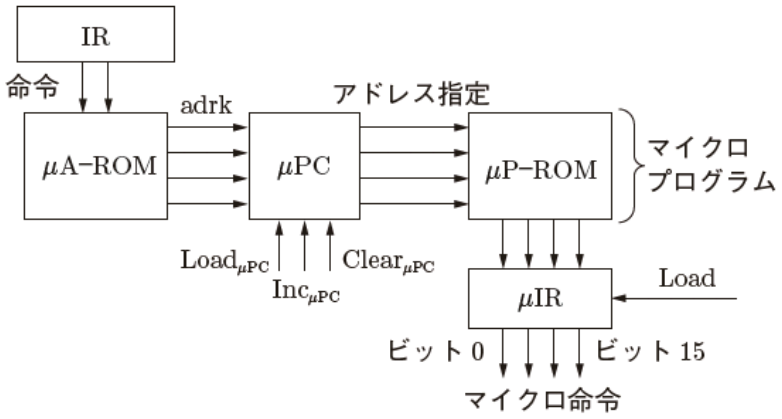
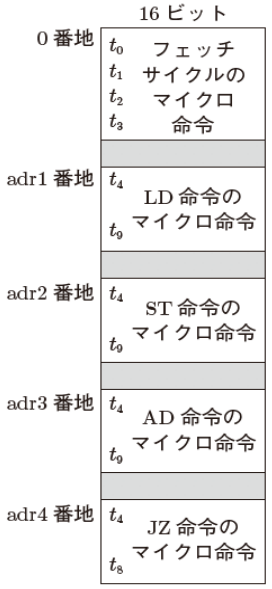


図 A・11 マイクロ命令のアドレス指定



（a） マイクロ命令の格納

ビット	信号
0	Inc_{PC}
1	$Load_{PC}$
2	$Gate_{PC}$
3	$Load_{IR}$
4	ALUMODE 0:PASS 1:ADD
5	$Load_{AC}$
6	$Gate_{AC}$
7	$Load_{MBR}$
8	$Load_{memory}$
9	$Gate_{MBR}$
10	$Load_{MAR}$
11	R/W 0:R 1:W
12	$Inc_{\mu PC}$
13	$Clear_{\mu PC}$
14	$Load_{\mu PC}$
15	$Test0_{AC}$

（b） マイクロ命令の形式

図 A・10 マイクロ命令の格納と形式

CPU の実現法
～μアドレスの指定～

- 各命令の最後のマイクロ命令を実行したとき，および “Test AC=0 ?” で AC = 0 でない場合は $Clear_{\mu PC}$ 信号によって μPC の値を 0 にする．また，フェッチサイクルの最後の状態では，各命令の実行サイクルのマイクロ命令の開始番地を $Load_{\mu PC}$ 信号によってロードすることで各命令の実行サイクルに遷移できる．例えば，サイクル t_0（MAR←PC）におけるマイクロ命令は，$Gate_{PC}$ 信号（ビット2）と $Load_{MAR}$ 信号（ビット10）および $Inc_{\mu PC}$ 信号 （ビット12）をオンに，残りのビットをオフにすればよい．よって，0 番地の 16 ビットのマイクロ命令（ビット0, 1, 2,…, 15）は次のようになる．

0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0

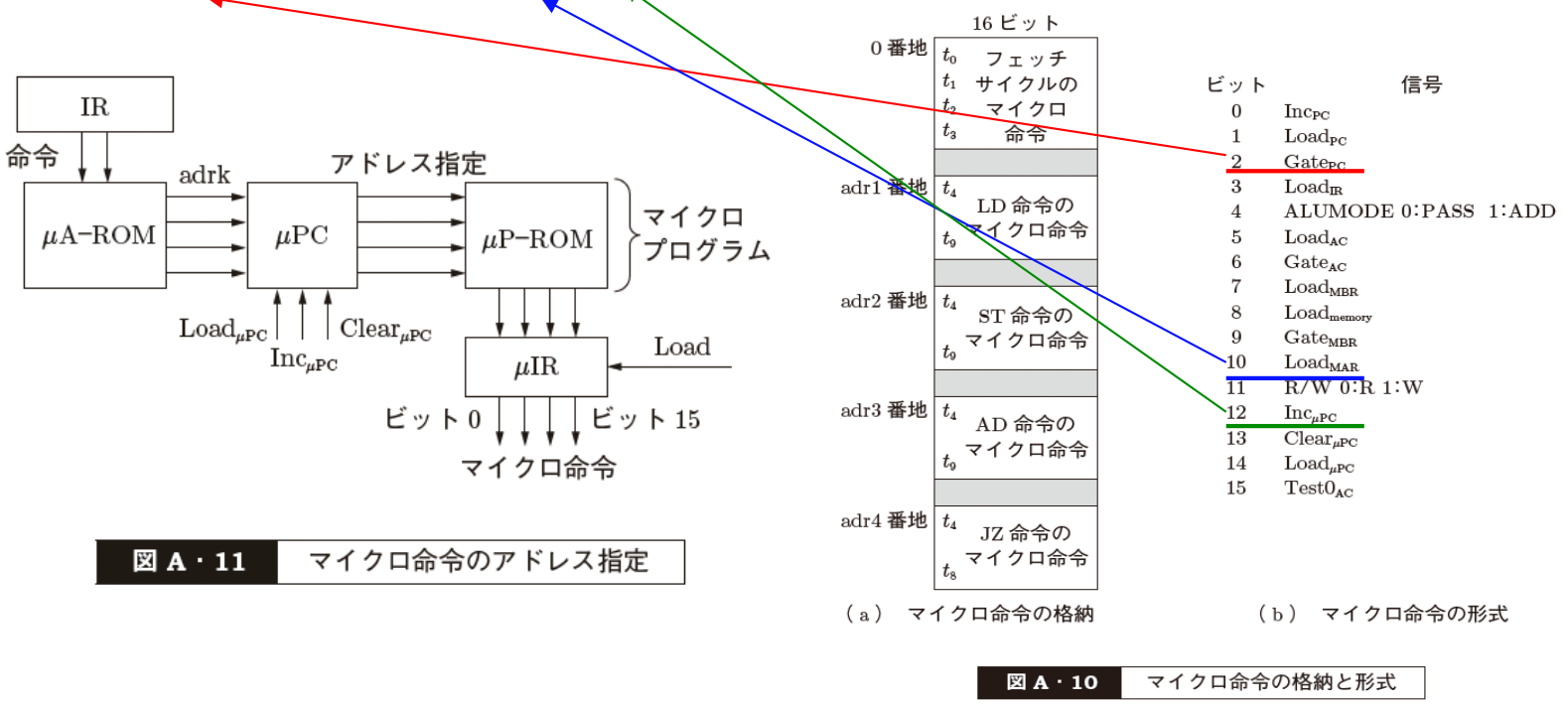
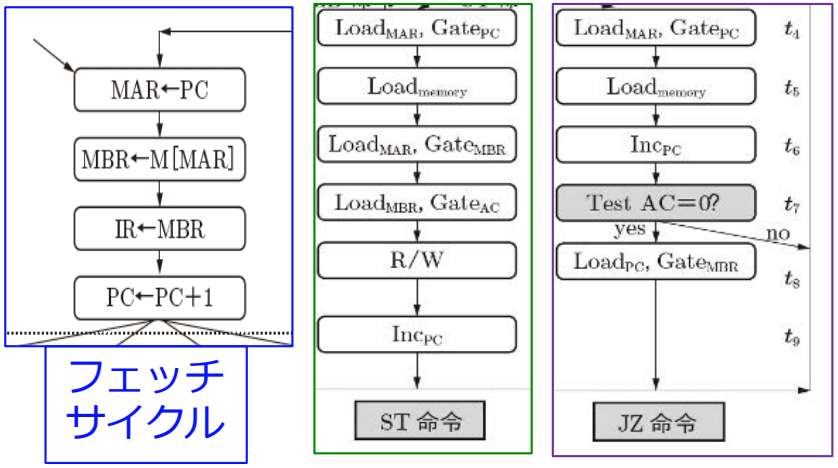


図 A・11 マイクロ命令のアドレス指定

図 A・10 マイクロ命令の格納と形式

CPU の実現法
～μアドレスの指定～

- なお，条件分岐 “Test AC=0 ?” を実行するときのみ，$Test0_{AC}$ 信号（ビット15）をオンにする．$Test0_{AC}$ 信号がオンでかつレジスタAC の $Zero_{AC}$ 信号が オフ（$Zero_{AC}$ = 0）のとき，μPC の $Clear_{\mu PC}$ 信号をオンにし，μPC の値を 0 にする．（$Zero_{AC}$ = 1）のときは，μPC の $Inc_{\mu PC}$ 信号をオンにし，μPC の値を 1 増加させる．他の番地のマイクロ命令については 0 番地のマイクロ命令と同様の方法で決定すればよい．図A·12 にフェッチサイクルの μP-ROM の内容と，ST 命令，JZ 命令の μP-ROM の内容を示す．



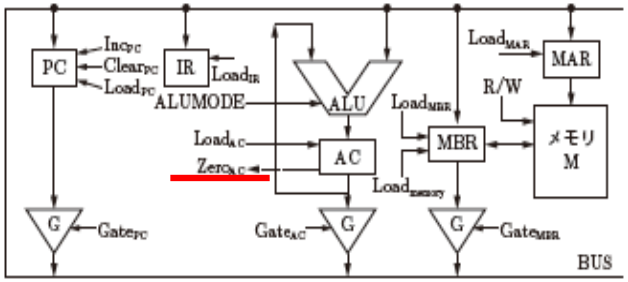


図 A・5　BUS を用いた CPU の回路図

ビット	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
マイクロ命令	Inc_{PC}	$Load_{PC}$	$Gate_{PC}$	$Load_{IR}$	ALUMODE	$Load_{AC}$	$Gate_{AC}$	$Load_{MBR}$	$Load_{memory}$	$Gate_{MBR}$	$Load_{MAR}$	R/W	$Inc_{\mu PC}$	$Clear_{\mu PC}$	$Load_{\mu PC}$	$Test0_{AC}$
0 番地	MAR ← PC	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0
1 番地	MBR ← M[MAR]	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
2 番地	IR ← MBR	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0
3 番地	PC ← PC +1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
……																	
adr2 番地	MAR ← PC	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0
(adr2 + 1)番地	MBR ← M[MAR]	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
(adr2 + 2)番地	MAR ← MBR	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0
(adr2 + 3)番地	MBR ← AC	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0
(adr2 + 4)番地	M[MAR]← MBR	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
(adr2 + 5)番地	PC ← PC + 1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
……																	
adr4 番地	MAR ← PC	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0
(adr4 + 1)番地	MBR ← M[MAR]	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
(adr4 + 2)番地	PC ← PC + 1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
(adr4 + 3)番地	TestAC = 0?	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
(adr4 + 4)番地	PC ← MBR	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0

図 A・12　μP-ROM の内容

OSAKA UNIVERSITY

CPU の実現法
～μアドレスの指定～

- なお，条件分岐 “Test AC=0 ?” を実行するときのみ，$Test0_{AC}$ 信号（ビット15）をオンにする．$Test0_{AC}$ 信号がオンでかつレジスタAC の $Zero_{AC}$ 信号が オフ（$Zero_{AC}$ = 0）のとき，μPC の $Clear_{\mu PC}$ 信号をオンにし，μPC の値を 0 にする．（$Zero_{AC}$ = 1）のときは，μPC の $Inc_{\mu PC}$ 信号をオンにし，μPC の値を 1 増加させる．他の番地のマイクロ命令については 0 番地のマイクロ命令と同様の方法で決定すればよい．図A·12 にフェッチサイクルの μP-ROM の内容と，ST 命令，JZ 命令の μP-ROM の内容を示す．

（上記を具体的に実現するには）

- ビット14の$Load_{\mu PC}$ 信号はそのまま利用可能
- ビット13の $Clear_{\mu PC}$ 信号についてはそのまま利用せず，ビット15の $Test0_{AC}$ 信号を用いて，

  $Clear_{\mu PC} \vee Test0_{AC} \cdot \neg Zero_{AC}$

  に変更して， μPC の値を 0 にする．
- また，ビット12の $Inc_{\mu PC}$ についても，

  $Inc_{\mu PC} \vee Test0_{AC} \cdot Zero_{AC}$

  に変更して， μPC の値を 1 増加させる．

ビット	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
マイクロ命令	Inc_{PC}	$Load_{PC}$	$Gate_{PC}$	$Load_{IR}$	ALUMODE	$Load_{AC}$	$Gate_{AC}$	$Load_{MBR}$	$Load_{memory}$	$Gate_{MBR}$	$Load_{MAR}$	R/W	$Inc_{\mu PC}$	$Clear_{\mu PC}$	$Load_{\mu PC}$	$Test0_{AC}$
0 番地	MAR ← PC	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0
1 番地	MBR ← M[MAR]	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
2 番地	IR ← MBR	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0
3 番地	PC ← PC +1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
……																	
adr2 番地	MAR ← PC	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0
(adr2 + 1)番地	MBR ← M[MAR]	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
(adr2 + 2)番地	MAR ← MBR	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0
(adr2 + 3)番地	MBR ← AC	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0
(adr2 + 4)番地	M[MAR] ← MBR	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
(adr2 + 5)番地	PC ← PC + 1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
……																	
adr4 番地	MAR ← PC	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0
(adr4 + 1)番地	MBR ← M[MAR]	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
(adr4 + 2)番地	PC ← PC + 1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
(adr4 + 3)番地	TestAC = 0?	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
(adr4 + 4)番地	PC ← MBR	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0

図 A・12　μP-ROM の内容

初期化と入出力

- 計算機の実行開始時には，プログラムカウンタPC の値を実行開始番地0（本章では 0 番地から実行が開始されると仮定）に初期化しなければならない．また， μPC の値も 0 に初期化しなければならない．
  - これらの操作は，PC および μPC に Clear 信号を送ることにより，実行される．
- ここでは，特に入出力のための命令や部品を考慮していない．入出力はメモリの特定の番地への書き込み，読み出しとして実現しても良い（他の実現方法もある）．
  - 例えば，入力をディップスイッチで，出力を LED（発光ダイオードまたは 7 セグLED）などで実現すればよい．

付録 CPUの設計
演習問題

演習問題

1 図 A·12 の ST 命令，JZ 命令の μP-ROM の内容を参考に，LD 命令，AD 命令の μP-ROM の内容を記載せよ．

2 本章の CPU の実装法を参考に，適当な CPU の命令語を設定し，その CPU を実現する同期式順序回路を作成せよ．

OSAKA UNIVERSITY

演習問題 1

- 図A·12 の ST 命令，JZ 命令の μP-ROM の内容を参考に，LD 命令，AD 命令 の μP-ROM の内容を記載せよ.

ビット		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	マイクロ命令	Inc_{PC}	$Load_{PC}$	$Gate_{PC}$	$Load_{IR}$	ALUMODE	$Load_{AC}$	$Gate_{AC}$	$Load_{MBR}$	$Load_{memory}$	$Gate_{MBR}$	$Load_{MAR}$	R/W	$Inc_{\mu PC}$	$Clear_{\mu PC}$	$Load_{\mu PC}$	$Test0_{AC}$
0 番地	MAR ← PC	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0
1 番地	MBR ← M[MAR]	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
2 番地	IR ← MBR	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0
3 番地	PC ← PC +1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
……																	
adr2 番地	MAR ← PC	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0
(adr2 + 1)番地	MBR ← M[MAR]	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
(adr2 + 2)番地	MAR ← MBR	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0
(adr2 + 3)番地	MBR ← AC	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0
(adr2 + 4)番地	M[MAR]← MBR	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
(adr2 + 5)番地	PC ← PC + 1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
……																	
adr4 番地	MAR ← PC	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0
(adr4 + 1)番地	MBR ← M[MAR]	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
(adr4 + 2)番地	PC ← PC + 1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
(adr4 + 3)番地	TestAC = 0?	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
(adr4 + 4)番地	PC ← MBR	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0

図 A・12　μP-ROM の内容

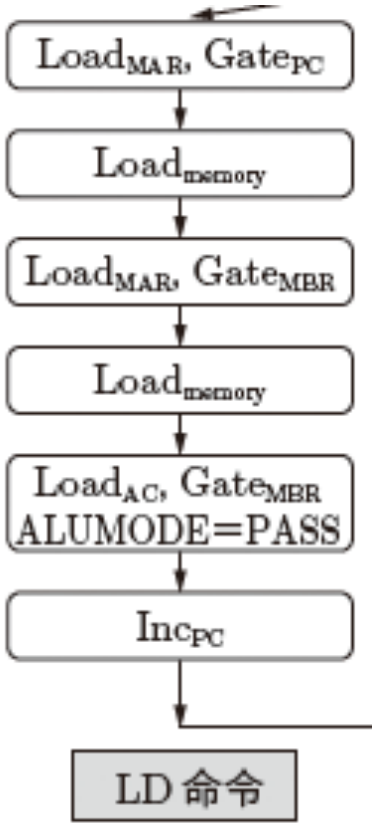
考慮時間

- 5分間程度で問題を解いてみてください．その間，ビデオを止めてください.

- この頁は30秒程度で次の頁に移行します.

演習問題１解答

- 図A·12 の ST 命令，JZ 命令の μP-ROM の内容を参考に，LD 命令，AD 命令 の μP-ROM の内容を記載せよ.

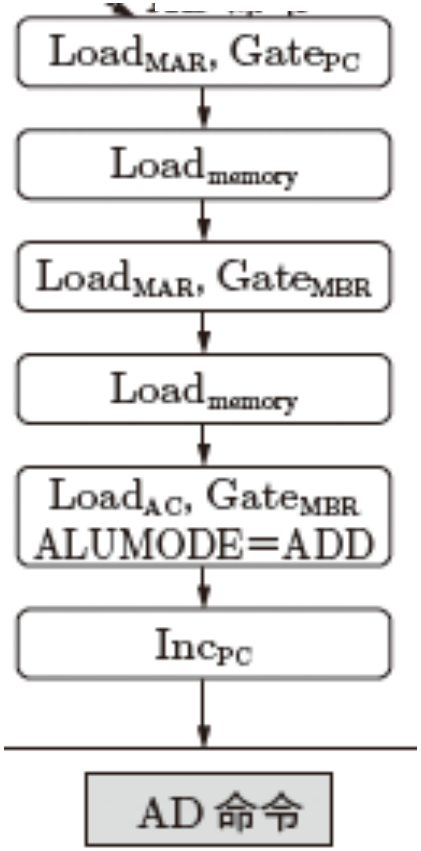


ビット		0	1	2	3	4	5	6	7	8	9	10	11	12	13	11	15
	マイクロ命令	Inc_{PC}	$Load_{PC}$	$Gate_{PC}$	$Load_{IR}$	ALUMODE	$Load_{AC}$	$Gate_{AC}$	$Load_{MBR}$	$Load_{memory}$	$Gate_{MBR}$	$Load_{MAR}$	R/W	$Inc_{\mu PC}$	$Clear_{\mu PC}$	$Load_{\mu PC}$	$Test0_{AC}$
adr1 番地	MAR←PC	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0
(adr1＋1)番地	MBR←M[MAR]	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
(adr1＋2)番地	MAR←MBR	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0
(adr1＋3)番地	MBR←M[MAR]	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
(adr1＋4)番地	AC←MBR	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0
(adr1＋5)番地	PC←PC＋1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
……																	
adr3 番地	MAR←PC	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0
(adr3＋1)番地	MBR←M[MAR]	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
(adr3＋2)番地	MAR←MBR	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0
(adr3＋3)番地	MBR←M[MAR]	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
(adr3＋4)番地	AC←AC＋MBR	0	0	0	0	1	1	0	0	0	1	0	0	1	0	0	0
(adr3＋5)番地	PC←PC＋1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

解図 A・1 LD 命令，AD 命令の μP-ROM の内容

演習問題１解答

- 図A·12 の ST 命令，JZ 命令の μP-ROM の内容を参考に，LD 命令，AD 命令 の μP-ROM の内容を記載せよ.



ビット	0	1	2	3	4	5	6	7	8	9	10	11	12	13	11	15
	マイクロ命令	Inc_{PC}	$Load_{PC}$	$Gate_{PC}$	$Load_{IR}$	ALUMODE	$Load_{AC}$	$Gate_{AC}$	$Load_{MBR}$	$Load_{memory}$	$Gate_{MBR}$	$Load_{MAR}$	R/W	$Inc_{\mu PC}$	$Clear_{\mu PC}$	$Load_{\mu PC}$	$Test0_{AC}$
adr1 番地	MAR←PC	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0
(adr1＋1)番地	MBR←M[MAR]	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
(adr1＋2)番地	MAR←MBR	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0
(adr1＋3)番地	MBR←M[MAR]	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
(adr1＋4)番地	AC←MBR	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0
(adr1＋5)番地	PC←PC＋1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
……																	
adr3 番地	MAR←PC	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0
(adr3＋1)番地	MBR←M[MAR]	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
(adr3＋2)番地	MAR←MBR	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0
(adr3＋3)番地	MBR←M[MAR]	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
(adr3＋4)番地	AC←AC＋MBR	0	0	0	0	1	1	0	0	0	1	0	0	1	0	0	0
(adr3＋5)番地	PC←PC＋1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

解図 A・1　LD 命令，AD 命令の μP-ROM の内容

演習問題２＆解答例

- 本章の CPU の実装法を参考に，適当な CPU の命令語を設定し，その CPU を実現する同期式順序回路を作成せよ.

（解答例）

- 例えば，下記のように AC に左に 1 ビットシフトする（AC の値を 2 倍にする）ための信号 L-shift$_{AC}$ を追加すれば，AC の値を 2 倍にする命令（AC←2*AC）を下右図のような実行サイクルを追加すれば実現できる（マイクロ命令の操作列と対応する制御信号の値を併記）.

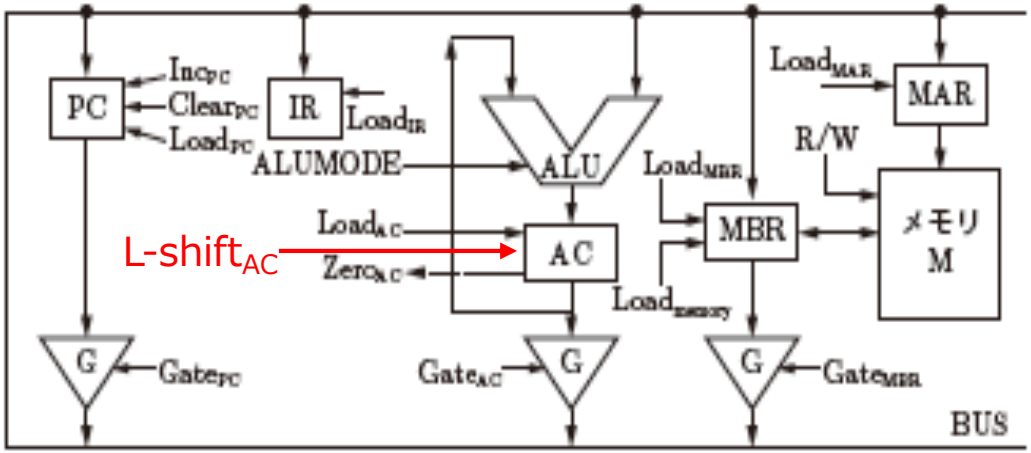
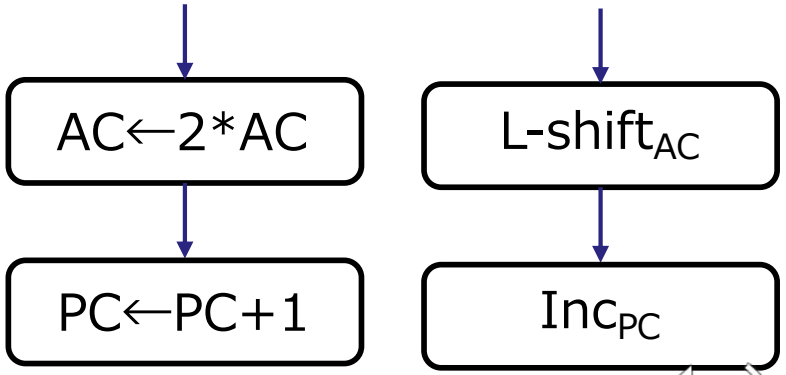


図 A・5 BUS を用いた CPU の回路図



本日のレポート課題

少し異なる回路のCPU設計

- 付録のCPUと少し異なる下記の図１の回路で構成されるCPUを設計したい．この回路のCPUについて，下記の各問に答えよ．
- 解答はCLEにアップする解答用紙を印刷して記入するか，同様の様式で各自の解答を記入して，CLEにアップしてください．

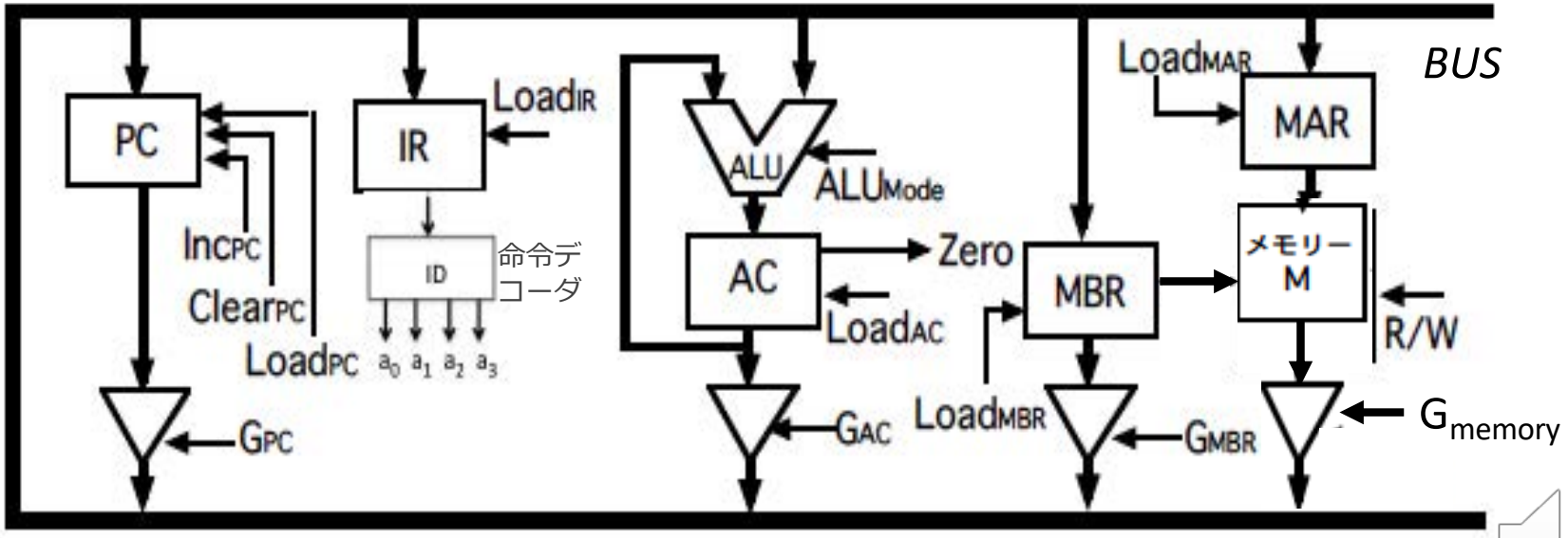


図１　CPUの回路構成

少し異なる回路のCPU設計

- この回路の各部品の動作は下記のとおり.
  - PC: プログラムカウンタ，Inc_{PC}=1のとき PC←PC+1， $Clear_{PC}$=1のとき PC←0，$Load_{PC}$=1のとき BUS 出力を PC にロードする（PC←AC, PC←MBR, PC←M[MAR]）
  - IR: 命令レジスタ，$Load_{IR}$=1のとき BUS 出力を IR にロードする．IRの命令に依存して命令デコーダ ID の a_0,a_1,a_2,a_3 信号のいずれかが 1 になる.
  - MAR: メモリーアドレスレジスタ，$Load_{MAR}$=1のとき BUS 出力を MAR にロードする.
  - MBR: メモリーバッファレジスタ，$Load_{MBR}$=1のとき BUS 出力を MBR にロードする.

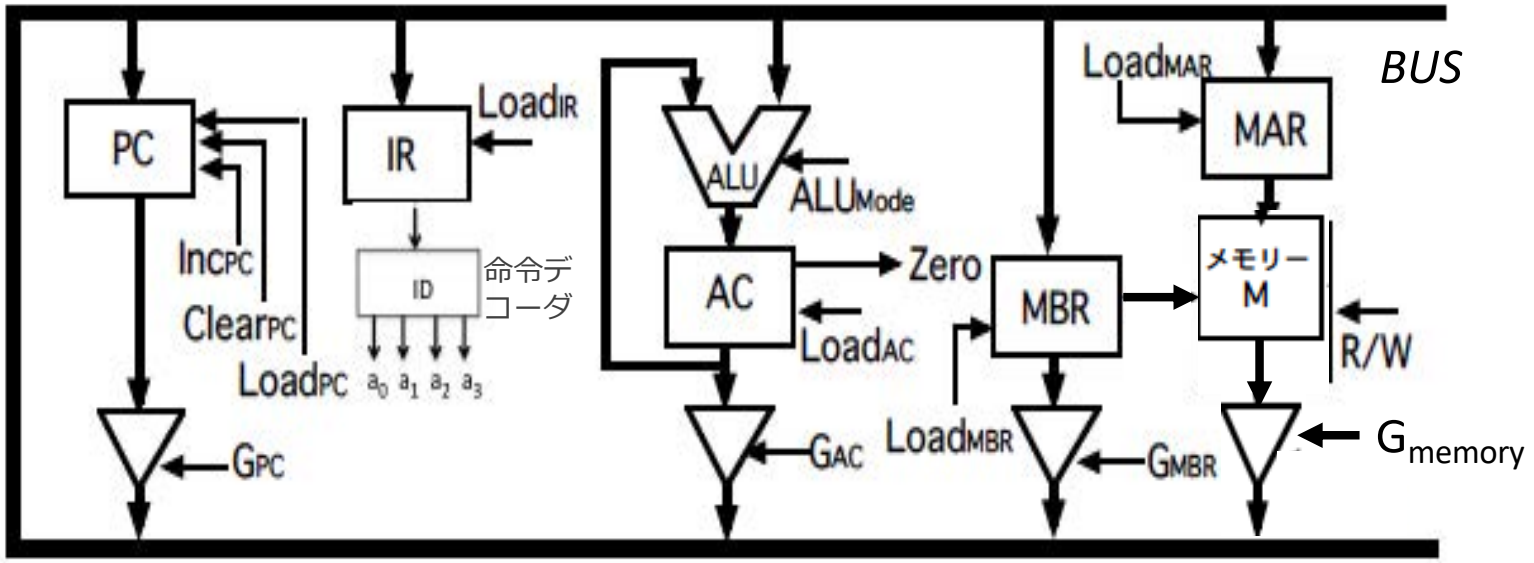
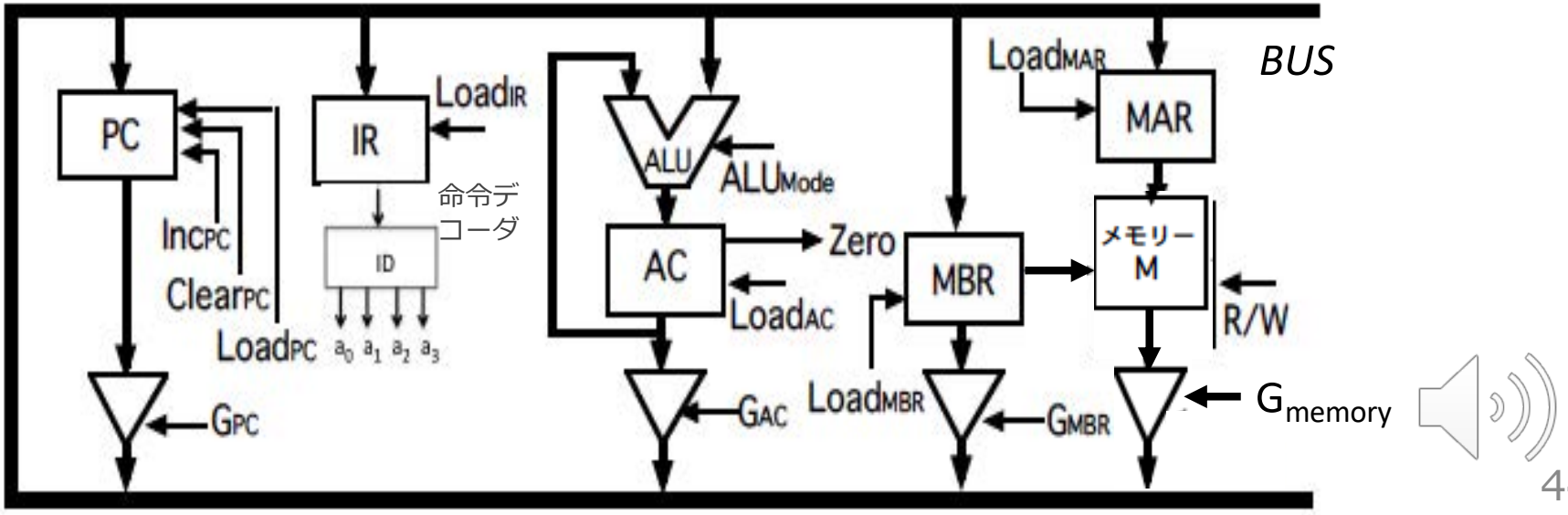


図１　CPUの回路構成

少し異なる回路のCPU設計

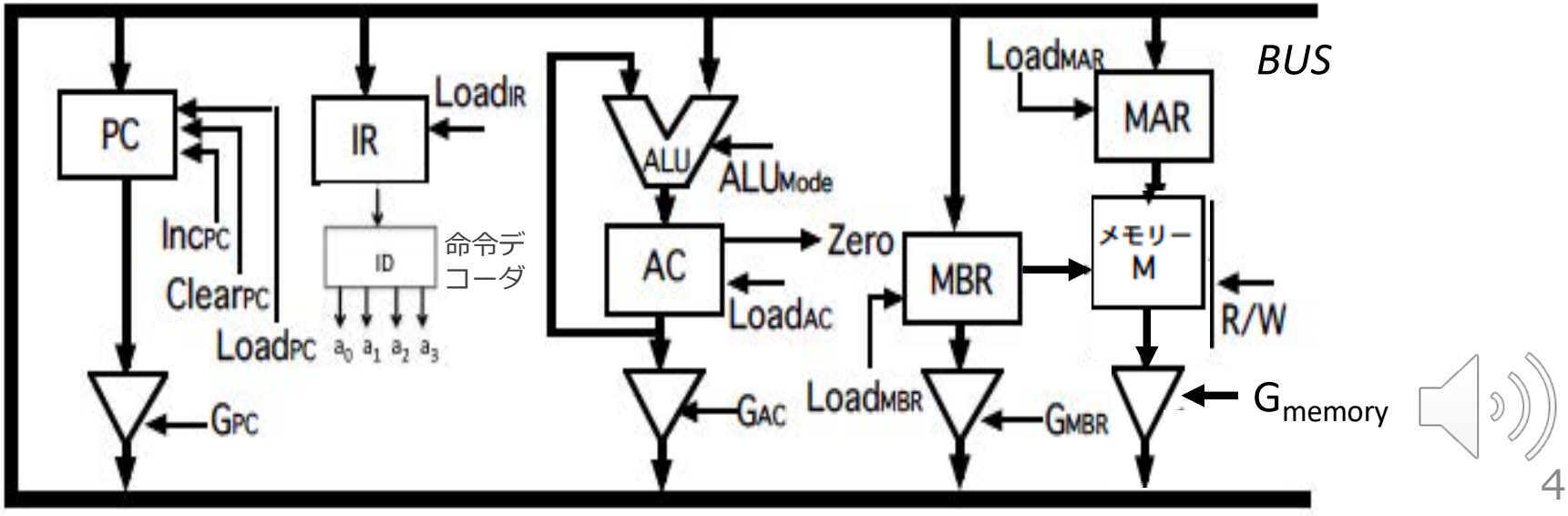
- この回路の各部品の動作は下記のとおり.
  - M: メモリー，R/W=0 のとき，メモリーMのMAR番地の内容 M[MAR] が下のG_{memory}側に出力される．R/W=1 のとき，MBRの内容がメモリーMのMAR番地に格納される（M[MAR]←MBR）.
  - AC: 汎用レジスタ，$Load_{AC}$=1 のとき ALU 出力を AC にロードする．AC=0 のとき，Zero信号が 1 になる（AC= 0 かどうかは「Test(AC=0)? 」で判定する）.
  - ALU: 算術論理演算装置，ALU_{Mode}=1 且つ G_{memory} =1 のとき AC←AC+M[MAR]，ALU_{Mode}=1 且つ G_{MBR} =1 のとき AC←AC+MBR，ALU_{Mode}=0 且つ G_{memory} =1 のとき AC←M[MAR]，ALU_{Mode}=0 且つ G_{MBR} =1 のとき AC←MBR（いずれも $Load_{AC}$=1 にした場合）



少し異なる回路のCPU設計

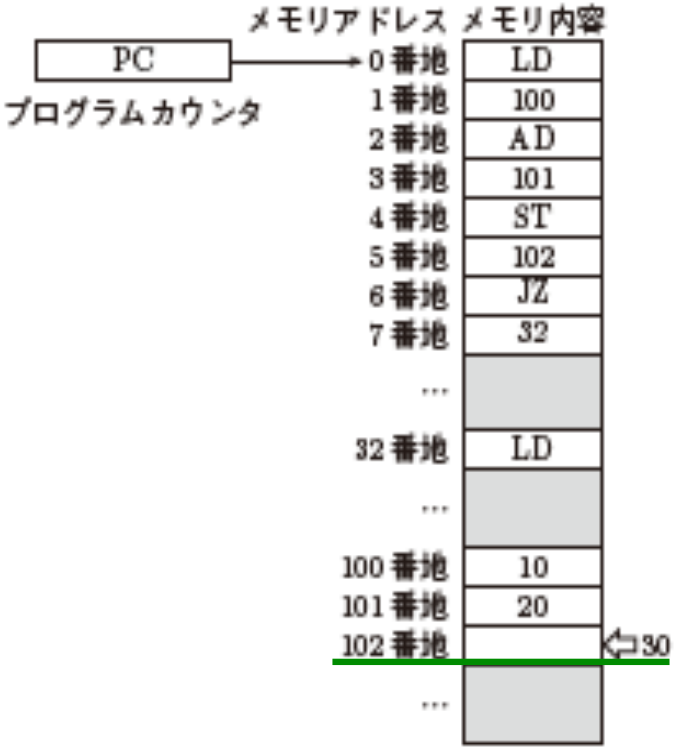
- このCPUの命令は下記の 4 命令のみで，1 語は 8 ビットで，各命令は命令コード，オペランドの順に格納された 2 語とする．レジスタは 8 ビットで，メモリーは 256 バイトとする．また，命令レジスタIRの命令デコーダ ID のa_0,a_1,a_2,a_3 信号の値は下記の通り．

命令コード	記号	オペランド	処理	IDの出力
00000000	LD	ADR	AC <- M[ADR] PC←PC+2	a0=1, a1=0, a2=0, a3=0
00000001	ST	ADR	M[ADR] <- AC PC←PC+2	a0=0, a1=1, a2=0, a3=0
00000010	AD	ADR	AC <- AC + M[ADR] PC←PC+2	a0=0, a1=0, a2=1, a3=0
00000011	JZ	B	if AC=0 then PC<-B else PC <-PC + 2	a0=0, a1=0, a2=0, a3=1



メモリに格納された機械語プログラム
（テキスト210頁のアセンブラと同じ）

- 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）

少し異なる回路のCPU設計

- このCPUのLD命令のフェッチサイクル，実行サイクルのマイクロ操作列は下記の通り．

	時刻	マイクロ操作
– フェッチサイクル	t0	MAR ← PC
	t1	IR ← M[MAR]
	t2	PC ← PC+1
– 実行サイクル	t3	MAR ← PC
	t4	MAR ← M[MAR]
	t5	AC ← M[MAR]
	t6	PC ← PC+1

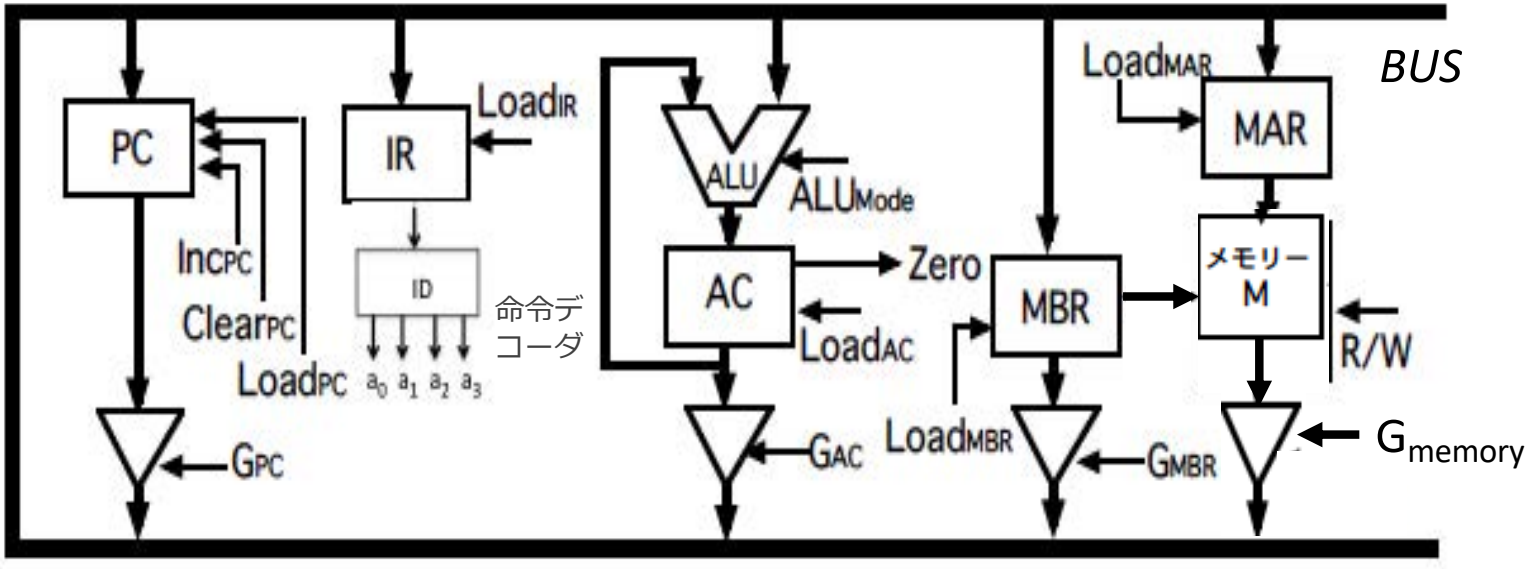
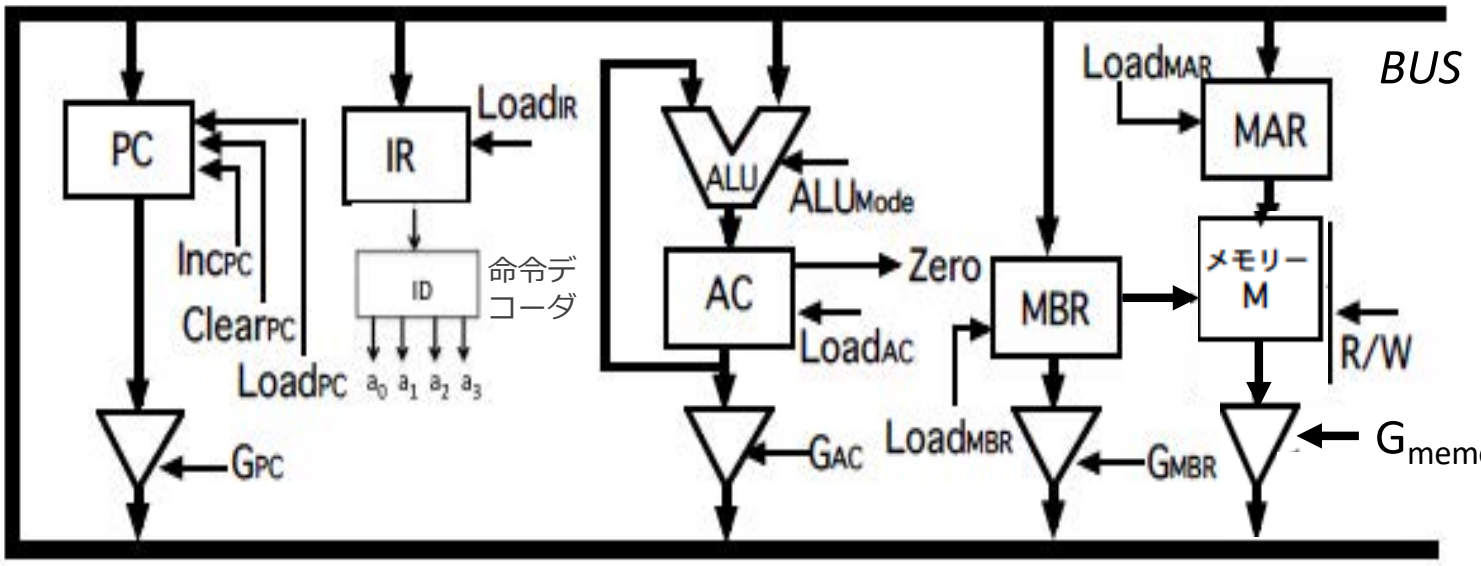


図１　CPUの回路構成

少し異なる回路のCPU設計

（問題１）

- 下記のLD命令の表記法にならって，ST, AD, JZの各命令の実行サイクルのマイクロ操作列を記載せよ（t0,t1,t2は共通のフェッチサイクル．t3以降の実行サイクルを記載せよ）．
  - フェッチサイクル　t0　MAR ← PC
    - t1　IR ← M[MAR]
    - t2　PC ← PC+1
  - 実行サイクル　t3　MAR ← PC
    - t4　MAR ← M[MAR]
    - t5　AC ← M[MAR]
    - t6　PC ← PC+1



少し異なる回路のCPU設計

（問題２）

- このCPU回路の制御部をマイクロプログラム方式で実現することを考える．マイクロ命令の形式は次のような17ビットのものとする．

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Inc_{PC}	$Clear_{PC}$	$Load_{PC}$	G_{PC}	$Load_{IR}$	ALU_{Mode}	$Load_{AC}$	G_{AC}	$Load_{MBR}$	G_{MBR}	$Load_{MAR}$	R/W	G_{Memory}	$Inc_{\mu PC}$	$Clr_{\mu PC}$	$Load_{\mu PC}$	Tp

このマイクロ命令は図2(a)のμP-ROMに格納する．μA-ROMは命令デコーダの値（命令xx）に対し，その命令のマイクロ操作列が書かれたμP-ROMの開始番地Adr_xxが出力される（図2(b)参照）．

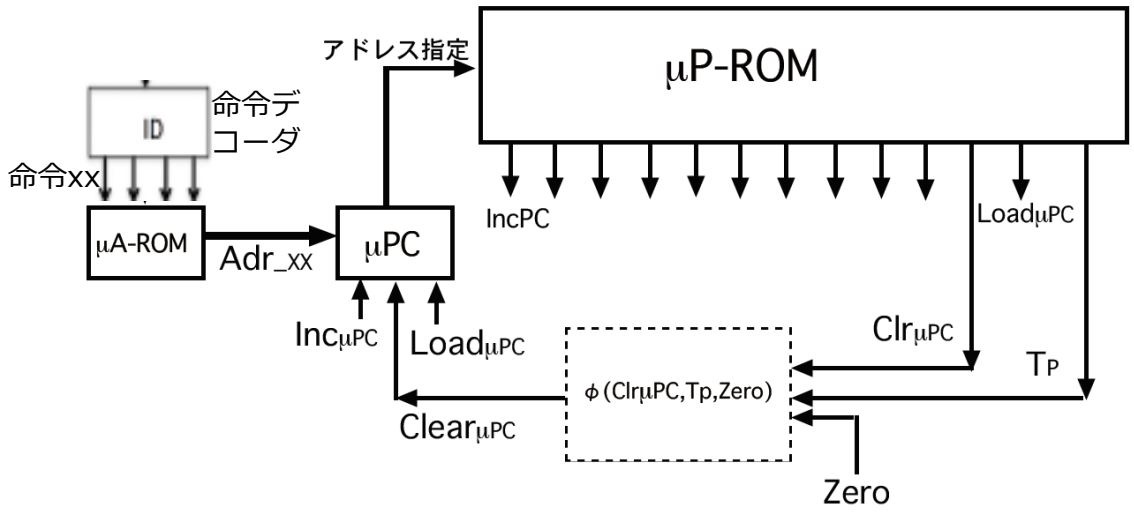


図２(a)：μP-ROMのアドレス指定

図２(b)：μP-ROMの内容

少し異なる回路のCPU設計

（問題２）

- $Clear_{\mu PC}$信号はμP-ROMの$Clr_{\mu PC}$信号（ビット15）とTp信号（JZ命令の分岐で利用する），ACの値が0 のとき 1 となるZero信号を用いた関数Φ($Clr_{\mu PC}$,Tp,Zero)で指定する．$Clr_{\mu PC}$信号は JZ命令 以外の命令の各実行サイクルの最後のマイクロ操作を実行した際に 1 にすることでμPCの値を 0 に戻す．
- マイクロプログラムカウンタμPCは$Clear_{\mu PC}$=1のとき μPC ← 0 となり，$Load_{\mu PC}$=1 のとき μPC ← Adr_xx となる．$Clear_{\mu PC}$=0 且つ $Inc_{\mu PC}$=1 のとき μPC ← μPC+1 となる．
- 教科書のJZ命令のマイクロ操作列と同様，¬(AC=0) のときに μPC の値を 0 に戻す場合，Tp の値を 1 にして Tp∧¬Zero が 1 になるときに μPC の値が 0 になるように指定する．

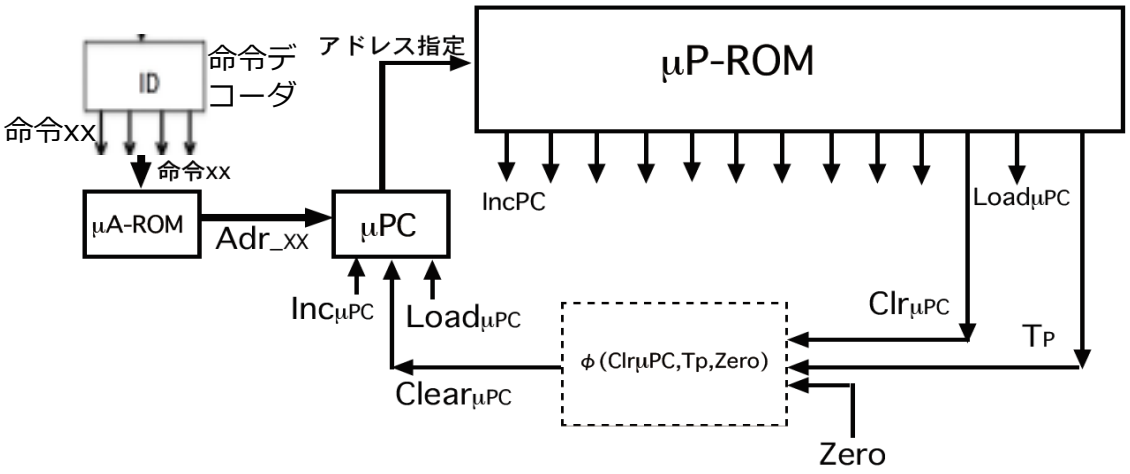


図２(a)：μP-ROMのアドレス指定

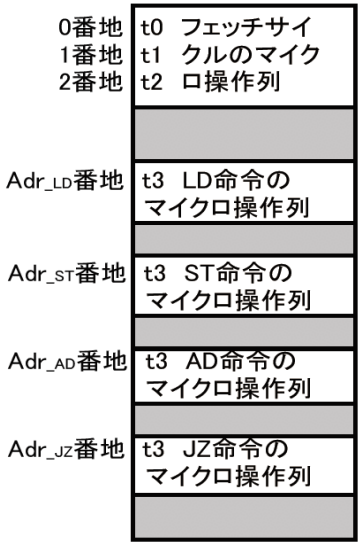


図２(b)：μP-ROMの内容

少し異なる回路のCPU設計

（問題２）

- この μP-ROM の 0 番地から 2 番地の内容と，ST，AD, JZ 命令のマイクロ操作列（17ビットの命令の列）の内容を下記のような表形式で作成せよ．表中の各セルには，0, 1, ×（ドント・ケア）のいずれかを記入すること.

（問題３）

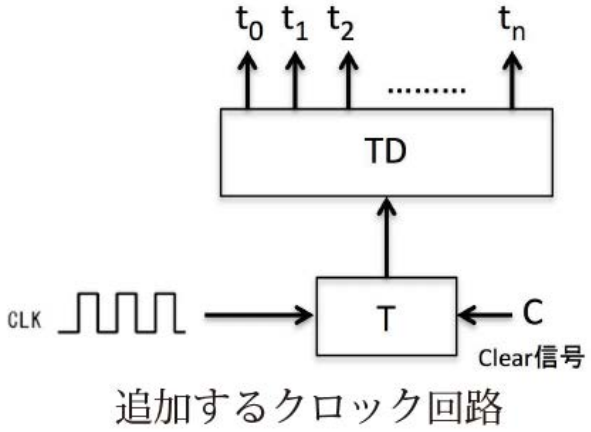
- $Clear_{\mu PC}=\Phi(Clr_{\mu PC},Tp,Zero)$ の $\Phi(Clr_{\mu PC},Tp,Zero)$ の論理式を $Clr_{\mu PC},Tp,Zero$ からなる論理関数で表せ.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
番地	Inc$_{PC}$	Clear$_{PC}$	Load$_{PC}$	G$_{PC}$	Load$_{IR}$	ALU$_{Mode}$	Load$_{AC}$	G$_{AC}$	Load$_{MBR}$	G$_{MBR}$	Load$_{MAR}$	R/W	G$_{Memory}$	Inc$_{\mu PC}$	Clr$_{\mu PC}$	Load$_{\mu PC}$	Tp
0番地																	
1番地																	
2番地																	
	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
Adr_LD番地																	
Adr_LD+1番地																	
Adr_LD+2番地																	
Adr_LD+3番地																	

少し異なる回路のCPU設計

（問題４）

- 図2(a)の回路に右図のクロック回路（時刻 t_i にTDの t_i 信号が 1 になる）を加えた場合に，信号線 Inc_{PC}, $Clear_{PC}$ の論理式をそれぞれ $t_0,t_1,\cdots,t_n$, a_0,a_1,a_2,a_3,Zeroなどの論理式で表せ.
- なおクロック T の Clear信号 C は $Clear_{\mu PC}$ を用いるものとする.



追加するクロック回路

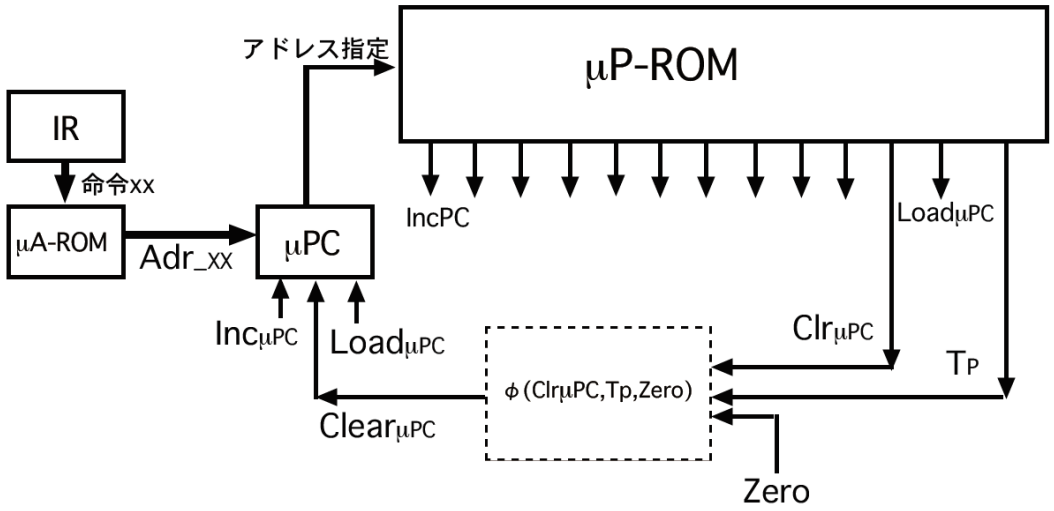


図２(a)：μP-ROMのアドレス指定

図２(b)：μP-ROMの内容

解答用紙

論理設計演習問題 2021 解答用紙

(1) ST, AD, JZ 命令マイクロ操作列は？

	LD	ST	AD	JZ
t3	MAR ← PC			
t4	MAR ← M[MAR]			
t5	AC ← M[MAR]			
t6	PC ← PC+1			
t7				

(2) μP-ROM の内容は?

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
マイクロ操作列	番地	IncPC	ClearPC	LoadPC	GPC	LoadIR	ALUMode	LoadAC	GAC	LoadMBR	GMBR	LoadMAR	R/W	GMemory	IncμPC	ClrμPC	LoadμPC	Tp
MAR ← PC	0番地																	
IR ← M[MAR]	1番地																	
PC ← PC+1	2番地																	
		:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
MAR ← PC	Adr_LD番地																	
MAR ← M[MAR]	Adr_LD+1番地																	
AC ← M[MAR]	Adr_LD+2番地																	
PC ← PC+1	Adr_LD+3番地																	
		:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
	Adr_ST番地																	
	Adr_ST+1番地																	
	Adr_ST+2番地																	
	Adr_ST+3番地																	
	Adr_ST+4番地																	
		:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
	Adr_AD番地																	
	Adr_AD+1番地																	
	Adr_AD+2番地																	
	Adr_AD+3番地																	
		:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
	Adr_JZ番地																	
	Adr_JZ+1番地																	
	Adr_JZ+2番地																	
	Adr_JZ+3番地																	

(3) ClearμPC＝φ(ClrμPC,Tp,Zero)＝

(4) IncPC＝　　LoadPC＝

OSAKA UNIVERSITY

13回目の授業終了

授業終了

皆さん
レポート提出してくださいね！