計算機アーキテクチャ講義
ノート３
(P. 83～P. 108)

平成19年11月26日配布
今瀬　真

講義ノートは下記のURLからダウンロード可能
（PDF形式）
http://www.ispl.jp/~imase/lecture/comp-arch/2005/

3. 2. 1演算処理部と命令の実行

- レジスタと状態情報はフリップフロップの集合
- その他は組合せ論理回路
- 実行制御部：機械語命令を解読し、制御ゲート・、演算装置ALU、アドレス計算/変換回路（以下adrUと表記）を制御する制御信号の系列（1機械語命令を実現するのに複数のクロックタイムを必要とする）を生成する。

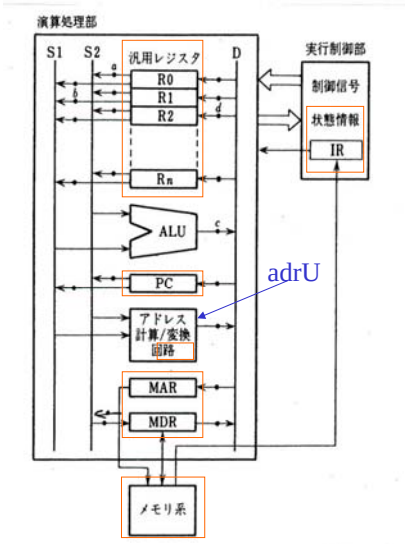


図 3.12　演算処理部と実行制御部（●印は制御ゲートの位置を示す）

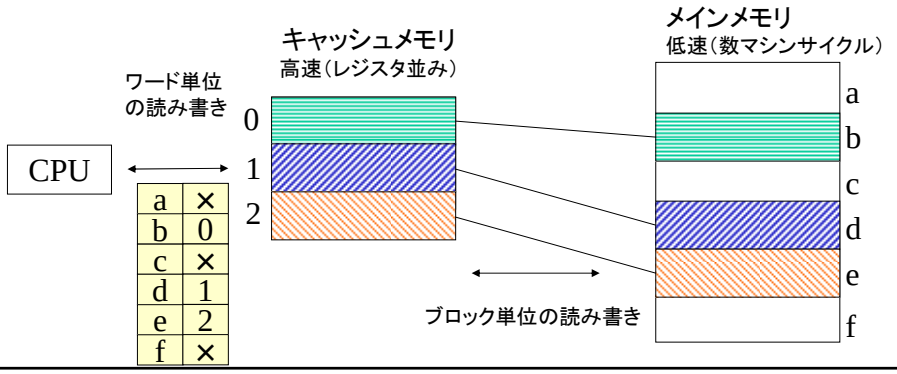
3.2.1演算処理部と命令の実行

1機械語命令が何クロックで実現できるかを考えてみる。簡単化のために以下を仮定する。

- すべての演算は1クロックで実現できるものとする。
- メモリへの書き込み、読み出しは1クロックで実現される。（キャッシュメモリを利用することによりある確率で実現可能である。詳細は後述(p.108)）。
- 仮想メモリアーキテクチャである。*Table*()という関数が仮想アドレスから実アドレスへの変換関数である。詳細は後述(p.144)。（仮想アドレス空間とは、計算機システムの利用者にとってアドレス可能な主記憶装置と見なしうる記憶空間である。実際の主記憶（実アドレス空間）はこれより小さい。仮想アドレス空間の内容はディスクに保存されており、主記憶には現在使用されているものだけが記憶されている。仮想アドレス空間から実アドレス空間への変換が必要であり、これを行うのがアドレス計算/変換回路adrU）

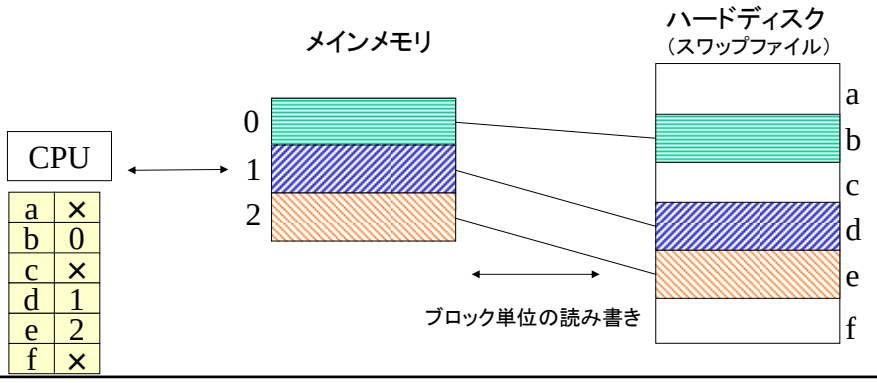
用語説明：キャッシュメモリ

- 一般的に、メインメモリに使われているDRAMのスピードはCPUに比べてかなり遅く、CPUの命令実行速度を下げる原因となっている。
- この問題を解決するために、CPUとメインメモリの間にキャッシュメモリと呼ばれる高速／小容量のメモリを配置する。
- メインメモリからキャッシュメモリへはブロック（数語から数十語単位）にまとめて転送する手法がとられる。
- キャッシュメモリにデータがある時は1クロックでデータの読み書きができる。ない場合は複数クロックかかる



用語説明：仮想メモリ

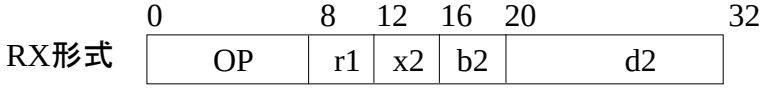
- ハードディスクをメインメモリの代用として利用するOSの機能。また、その機能を利用して確保された、実際のメモリ容量以上のメモリ領域。「仮想記憶」とも言う。ハードディスク上に「スワップファイル」と呼ばれる専用の領域を用意して、メモリ容量が不足してきたら使われていないメモリ領域の内容を一時的にハードディスクに退避させ、必要に応じてメモリに書き戻すことで実現される。
- キャッシュとよく似ているが、アドレス計算テーブルをソフトウェアで管理しており、そのテーブルはCPU内にある。



a	×
b	0
c	×
d	1
e	2
f	×

3.2.1演算処理部と命令の実行

System370のRX形式の加算命令の場合
r1←(r1)+memory[(x2)+(b2)+d2]



OP：操作符号
r1 ：汎用レジスタまたは浮動少数点レジスタを指定
x2 ：汎用レジスタ（インデックスレジスタ）を指定
b2 ：汎用レジスタ（ベースレジスタ）を指定
d2 ：ベースアドレスからの偏位

3. 2. 1演算処理部と命令の実行

System370のRX形式の加算命令

r1←(r1)+memory[(x2)+(b2)+d2]

1. 命令の読み出し(IF) :

IR、MDR←*Memory*(PC)

**PCとMARは直結されている。図3.12ではこの線は省略されている。

**PCには仮想アドレスでなく実アドレスが入っていると仮定する。

2. 命令の解読(ID):

命令の形式と語長を解読し、

命令の各部を抽出(OP、r1、x2、b2)。

PC←PC＋k　k=2,4,6 (RX形式の加算命令では4)

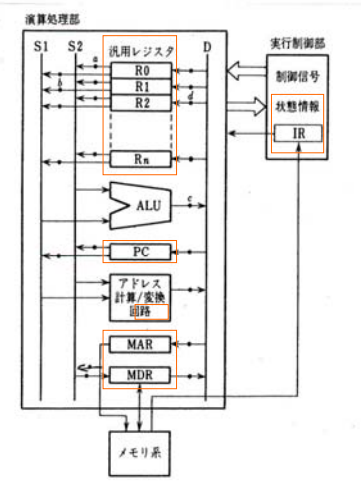


図 3.12　演算処理部と実行制御部（●印は制御ゲートの位置を示す）

3. 2. 1演算処理部と命令の実行

System370のRX形式の加算命令

r1←(r1)+memory[(x2)+(b2)+d2]

3. オペランド・アドレスの計算(AC)

VAR←(x2)+(b2)+$(MDR)_{20-32}$

**図3. 12ではMDRからadrUへの線は省略されている。

**$(MDR)_{20-32}$はd2であり本演算はadrU(アドレス計算/変換回路)で行われる。

**VARは仮想アドレスを保持するレジスタでadrU内にある。

4. アドレス変換(AT)

MAR←*Table*(VAR)　メモリ読み出し指令

5. オペランドの読み出し(OF)

MDR←*Memory*(MAR)

6. 演算の実行(EX)

r1←(r1)+(MDR)

**加算命令なので＋。演算の種類は、OP部で決まる。

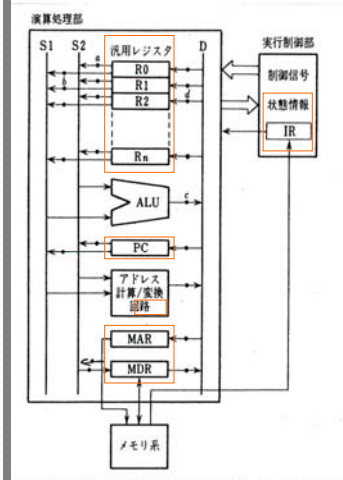


図 3.12　演算処理部と実行制御部（●印は制御ゲートの位置を示す）

3. 2. 1演算処理部と命令の実行

- 各機械語命令で実行に必要となるクロック数はことなる

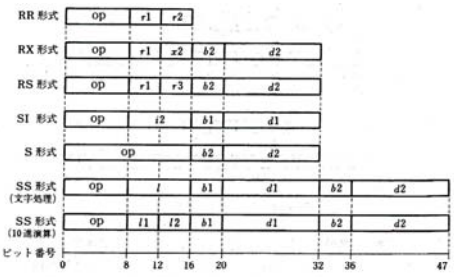


図 2.12 System 370 の命令語の形式

op：操作符号
r：汎用レジスタまたは浮動小数点レジスタを指定（オペランド）
x：汎用レジスタを指定（インデックス・レジスタ）
b：汎用レジスタを指定（ベース・レジスタ）
d：ベース・アドレスからの偏位
i：イミディエイト・オペランド
l：可変長オペランドの長さを指定

•RR形式　r1←(r1)*(r2)
　　IF→ID→EX
•RX形式　r1←(r1)*M[(x2)+(b2)+d2]
　　IF→ID→AC→AT→OF→EX
•SI形式　M[(b1)+d1] ←i2*M[(b1)+d1]
　　IF→ID→AC→AT→OF→EX'→MW
•ロード命令(RX形式 ）r1 ← M[(x2)+(b2)+d2]
　　IF→ ID→AC→AT→OF→EX''
•ストア命令(RX形式　M[(x2)+(b2)+d2] ← (r1)
　　IF→ID→AC→AT→MW
•分岐命令(S形式の一部） PC ←(b2)+d2
　　IF→ID→PC
RSとSS形式については省略
　　EX'：即値との論理演算
　　EX''：MDRからGRへの転送
　　PC：分岐条件の判定とPCの設定

3. 2. 1演算処理部と命令の実行

- 各機械語命令で実行に必要となるクロック数はことなる
- **CPI**：Clock Per Instruction 1命令あたりの平均所要クロック数。どの命令がどのような頻度でててくるかは、実測による。
- **MIPS**：CPI×クロックのサイクルタイム（秒）×１００万
  - CPIやMIPSは測定に使用するプログラムやデータなどに依存するため、値を比較するときに注意が必要。

表 3.3 モデル計算機の命令と実行ステップ

命令の実行ステップ	RR形式演算命令	RX形式演算命令	SI形式演算命令	ロード命令	ストア命令	分岐命令
1	IF	IF	IF	IF	IF	IF
2	ID	ID	ID	ID	ID	ID
3	EX	AC	AC	AC	AC	AC 分岐条件判定
4		AT	AT	AT	AT	
5		OF	OF	OF	MW	
6		EX	即値との論理演算	MDRからGRへ転送		
7			MW			

3.2.2 実行制御部の構成

（1）状態遷移図による表現（実行制御部の仕様）

- 状態遷移図：ある時点の状態に対するすべての入力と、その入力により遷移する次の状態の関係を表す。
- ここでは、入力でなく命令の種類となる。
  - ○：状態　→：命令の種類に対する次の遷移

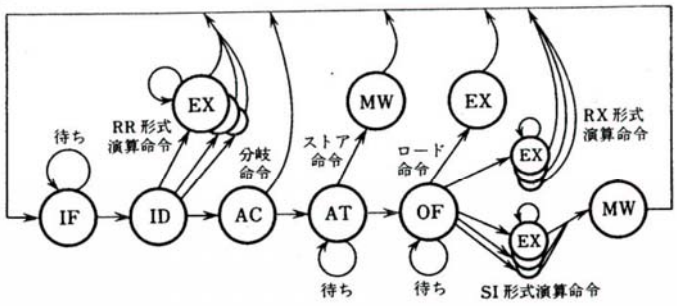


図 3.13　命令の実行制御の状態遷移図

3.2.2 実行制御部の構成

（1）状態遷移図による表現（実行制御部の仕様）

- 図中の自己ループは、メモリへのアクセスで待ちが生じた場合に通過する。
- RS形式とSS形式は省略。
- 演算命令単位に状態がある。
- 各状態に対応して、演算処理部への制御信号が定まる。
- 実行制御部を実現する方式として下記の2方式がある。
  - 布線論理制御方式
  - マイクロプログラミング方式がある。

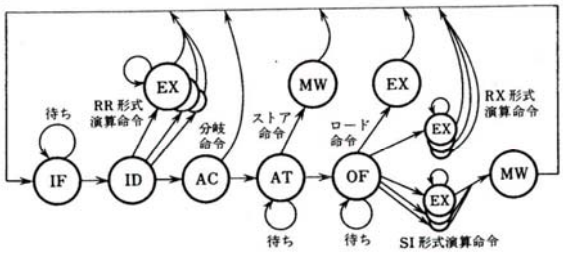
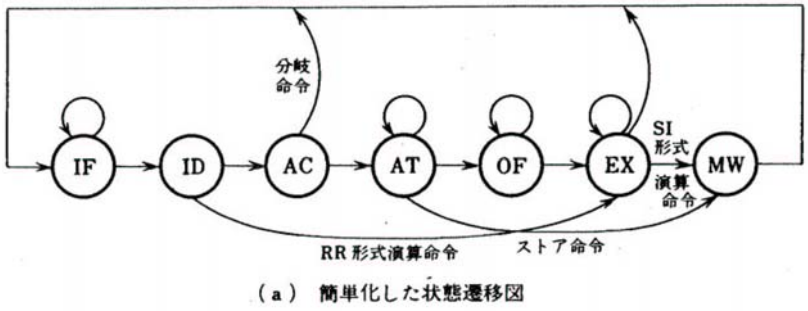


図 3.13　命令の実行制御の状態遷移図

3.2.2 実行制御部の構成

(2)実現方式1:布線論理制御方式

- 状態遷移図の簡単化(図3．14(a)):EXを共通化している。
- このためEXの実行ではALUへの制御信号は命令コードに依存する。



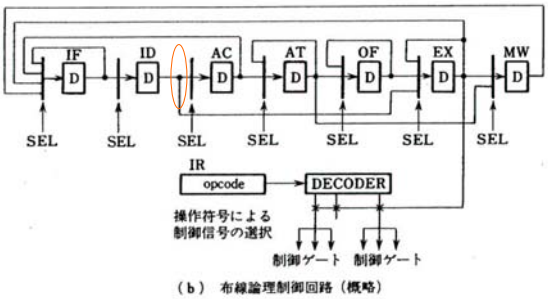
(a) 簡単化した状態遷移図

3.2.2 実行制御部の構成

(2)実現方式1:布線論理制御方式

- 布線論理回路(図3．14(b):Dフリップフロップ(図中のD)に1がたつ時に、状態遷移図の対応する状態にある。1クロックずつ1がたつ場所が移動する。
- 次のクロックでどこに1が立つかを制御するのがSELである(例参照)。
- SELの開閉は命令の種類と主記憶からの読み込み書き込み終了信号で決定される。
- DECODERは、SELや演算の実行を制御。命令(opcode)値とDフリップフロップの状態により、どの制御ゲートを開くかは一意にきまる。

例1(右から3つ目のSEL):
RR形式以外(RX形式かSI形式かロード命令かストア命令か分岐命令)の時に開く。
例2(右から4つ目のSEL):
分岐命令以外かメモリからの読み込み終了の時に開く

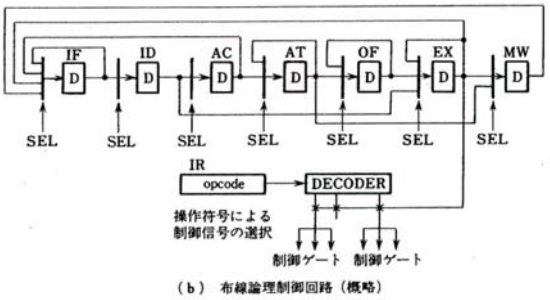


(b) 布線論理制御回路(概略)

3.2.2 実行制御部の構成

布線論理制御方式の欠点

- 複雑な命令を実現しようとすると状態が増える（例えば、SS形式命令）
- 論理回路はゲート数を少なくするための最適化をはかるので、1命令でも追加しようとすると新たに設計をしなおす必要がある。



（b） 布線論理制御回路（概略）

3.2.2 実行制御部の構成

（3）実現方式2：マイクプログラミング制御（図3．15）

- マイクロプログラムの1行が1クロックの動作を記述している。
- マイクロ命令：どのゲートを開けるかを記述
- 次命令アドレス：図3．14の状態遷移図のSELの役割（次にどの命令を実行するかを決定）

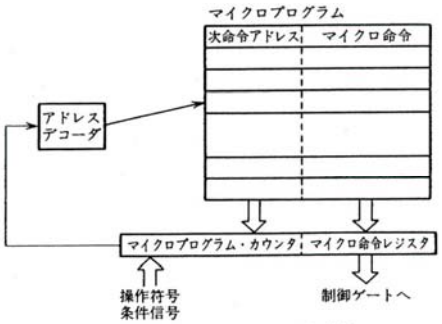


図 3.15 マイクロプログラム制御方式

3.2.2 実行制御部の構成

水平型マイクロプログラム方式:
- マイクロ命令の各ビットが演算処理部の各制御信号に1対1に対応。
- マイクロ命令のビット数を小さくするため、同時に1とならないものを符号化して表現(16本の制御信号があるとき、符号化すれば4ビットのマイクロ命令となる。)することもある(符号から制御信号への変換(デコーダ)による遅延がクロック幅を長くする必要がでてくる可能性がある。
- マイクロ命令の語長は100ビット程度、語数は1～2Kになる。

垂直型マイクロプログラム方式:
- 符号化をさらに進めて、レジスタ転送表現でマイクロ命令を記述(水平型では1クロックで記述できたことが、複数クロック必要となることがある。)。演算処理部が簡単な小型計算機に多い。

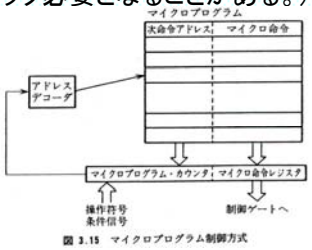


図 3.15 マイクロプログラム制御方式

3.2.2 実行制御部の構成

水平型プログラムでは、同じビットパターンのマイクロ命令が何箇所にも出現する。

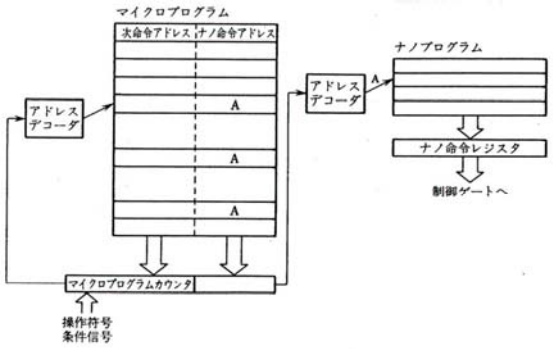


図 3.16 マイクロプログラムとナノプログラム

ナノプログラミング(図3．16):
- マイクロプログラムでは、マイクロ命令のアドレスを指定(ビットの開閉の指定はナノプログラムに記述)。
- 垂直型マイクロプログラム方式の1実現方式とも解釈できる。

3.2.2 実行制御部の構成

関連する用語

- ファームウェア：複数の機械後命令の系列が頻繁に使われる時、それらをまとめてマイクロプログラムで実現すること。
- エミュレーション：他の計算機の機械語の実行（模擬）をマイクロプログラム命令で実現する。新機種を開発するときに、従来機種の機械語ソフト資産を利用するために使われた技術。
- 仮想計算機：他の計算機の実行をOSレベルまで含めて模擬すること。

3.2.3命令実行のパイプライン制御

(1) パイプライン実行制御回路の構成

パイプライン制御：プロセッサの内部動作を機能ブロックの動作ステージに分割し，各ステージが互いに独立に動作するように構成する．各ブロックは入力を処理して次のステージのブロックへ結果を渡す．したがって，各ステージは並列に処理を実行し，命令がオーバーラップして実行されている．一つの命令実行時間は長くても，出口のブロックをみると短時間に命令が次々に処理されてくる．これをパイプライン制御という．

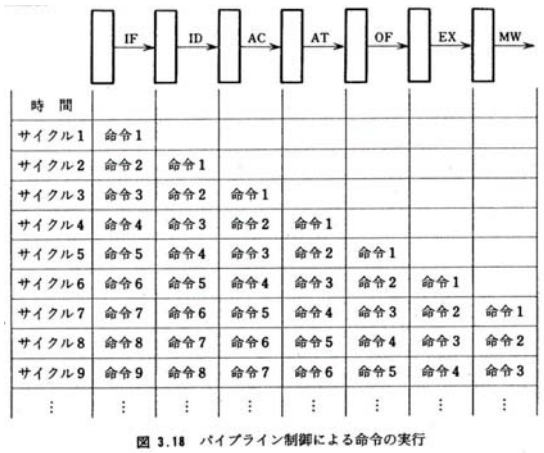


図 3.18 パイプライン制御による命令の実行

3.2.3命令実行のパイプライン制御

- 個々の機械語命令の実行時間（レイテンシー）は短くはならないが、単位時間あたりの機械語命令実行数（スループット）は短縮される。（図3．19参照）

命令＼時間	サイクル 1	2	3	4	5	6	7	8	9	…
1	IF	ID	AC	AT	OF	EX	MW			
2		IF	ID	AC	AT	OF	EX	MW		
3			IF	ID	AC	AT	OF	EX	MW	
4				IF	ID	AC	AT	OF	EX	…
5					IF	ID	AC	AT	OF	…
6						IF	ID	AC	AT	…
7							IF	ID	AC	…
8								IF	ID	…
9									IF	…
⋮										

図 3.19 命令に着目したパイプライン制御の表現法

3.2.3命令実行のパイプライン制御

(2) パイプラインの乱れ(ｉ) （多サイクルステージによる乱れ）
：あるステージの処理が、1サイクルで終了しない
原因1：メモリアクセスがキャッシュメモリにない場合（図3.20参照）
→解決策：キャッシュメモリのヒット率を向上させる。

•キャッシュメモリ：高速の小さなメモリ（1クロックで読み書き可能）
•主記憶の内容の一部を事前にキャッシュメモリにおいておき、通常はここに読み書きする。
•プログラムは近い領域を頻繁にアクセスする性質をもっているので、主記憶から1語でなくまとまった単位を一度にキャッシュに読み込むことにより、主記憶にアクセスする頻度が小さくなる。

	時間（サイクル） 5	6	7	8	9	10	…
命令1	OF	EX	MW				
2	AT	(OF)	(OF)	OF	EX	MW	
3	AC	(AT)	(AT)	AT	OF	EX	…
4	ID	(AC)	(AC)	AC	AT	OF	…
5	IF	(ID)	(ID)	ID	AC	AT	…
6		(IF)	(IF)	IF	ID	AC	…
7					IF	ID	…
8						IF	…

（b） 命令に着目した表現

3.2.3命令実行のパイプライン制御

(2) パイプラインの乱れ(ｉ) (多サイクルステージによる乱れ) :
原因2:乗算命令のように1サイクルで終了しない演算
→解決策:多サイクル演算回路の独立化(1サイクル演算と多サイクル演算を別に設ける)とパイプライン化(多サイクル演算器をパイプライン化する)(図3.21参照)。(iii)の問題が新たに発生することに注意

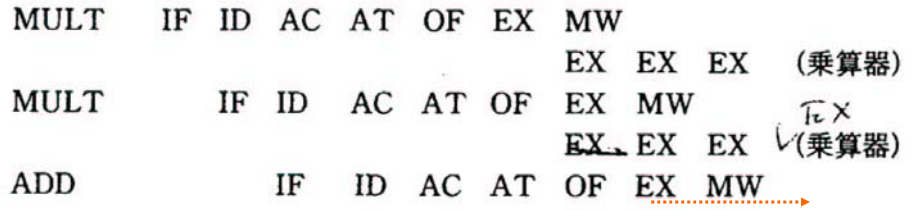


図 3.21 多サイクル演算に対する対処

3.2.3命令実行のパイプライン制御

(2) パイプラインの乱れ(ii) 機能回路要求競合による乱れ
原因1:メモリ系へのアクセス競合(IF, OF, MWフェーズ)図3.22参照
→解決策1:待たせてしまう(スループットが半分以上落ちる)
→解決策2:命令読み出しを一括して行う(命令は連続実行の確率が高いことを利用)
→解決策3:キャッシュメモリを命令用とデータ用の2つを用意

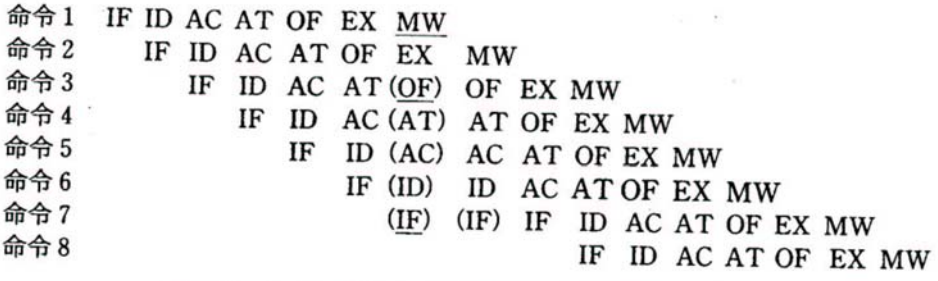


図 3.22 メモリ・アクセスの競合によるパイプラインの乱れ

3.2.3命令実行のパイプライン制御

(2) パイプラインの乱れ(ⅱ) 機能回路要求競合による乱れ
原因1:汎用レジスタへのアクセス競合(AC, EXフェーズ)
→解決策:バスの数を増やす。(ⅲ)の問題が残ることに注意

S1、S2、DはEXフェーズのみで利用し
あらたにS1'、S2'、D' を設けてこれはACフェーズのみで利用する。

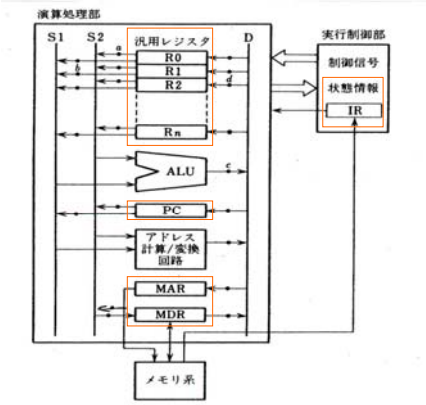


図 3.12 演算処理部と実行制御部(●印は制御ゲートの位置を示す)

3.2.3命令実行のパイプライン制御

(2) パイプラインの乱れ(ⅲ) 時間関係の逆転による乱れ
前の命令の実行結果を後の命令で使う

原因1:PCの加算(PCを幾つ加算すべきかは命令によって異なるので、IDフェズが終了してからでないと分からない。)
→解決策:IFフェーズで命令語長の解読とPCへの加算を行う。(命令語長のみを調べるのであれば簡単な回路で実現できることを利用。またPCの加算器を特別に作る。)

原因2:汎用レジスタへのアクセスの時間関係逆転
→解決策:待たせるしかない。本当に同じレジスタにアクセスした場合のみを検出する。IDフェーズでどのレジスタに書き込むか分かることを利用する。スコアボード(各レジスタに対応する1ビットのフラッグ)IDフェーズで書き込み予定が分かるとそのレジスタのスコアボードに1をたて、EXフェーズでクリアする。後続の命令が読み書きしようとする場合はスコアボードが1の間は待たせる。

3.2.3命令実行のパイプライン制御

(2) パイプラインの乱れ(iii) 時間関係の逆転による乱れ
　　前の命令の実行結果を後の命令で使う
原因3:多サイクル演算の独立化とパイプライン化
　→解決策:汎用レジスタへのアクセス競合に帰着
原因4:汎用レジスタへのアクセス競合(AC, EXフェーズ)のためのバスの多重化
　→解決策:汎用レジスタへのアクセス競合に帰着
原因5:メモリアクセスの時間関係逆転
　→解決策:汎用レジスタと本質的には同様だが、スコアボード監視と同様の監視はかなり複雑
原因6:分岐命令による乱れ(分岐命令のACが終了しないと次の命令のIFが実行不能)
　→解決策1:分岐がないと予測して、分岐があった場合は破棄
　→解決策2:前回その分岐命令を実行した時の飛び先を記憶(解決策1に比べて破棄する確率が減少)
　→解決策3:コンパイラーに任せる手法(遅延分岐)104頁の図参照

3.2.3命令実行のパイプライン制御

(3)ロード・ストア型計算機のパイプライン制御 (RISCマシン59頁参照)

- 命令セットがパイプライン制御を念頭において設計されておりパイプラインがCISC型に比べて簡潔に構成可能。
- RISCマシンのレジスタ構成:メモリ同様アドレス(レジスタ番号)を指定して読み書きする。読み出し回路が2つ、書き込み回路が1あり、1サイクルに同時に3つの動作が行える。

表 3.5　ロード・ストア型計算機の命令実行ステップ

	演算命令	ロード命令	ストア命令	分岐命令
IF	命令の読み出し，PC の更新			
ID	命令の解読，レジスタの読み出し			
EX	演算の実行	アドレス計算	アドレス計算 データを MDR へ転送	アドレス計算 分岐条件の判定
MA	——	メモリから読み出し	メモリへの書き込み	分岐成功の場合 PC を更新
WB	結果をレジスタに書き込み		——	——

3.2.3命令実行のパイプライン制御

(3)ロード・ストア型計算機のパイプライン制御

- 命令読み出し(IFフェーズ):命令語長が4バイトと固定であり、PC加算に語長の判定が不要
- 命令の解読(IDフェーズ):読み出しのレジスタ指定フィールドが一定(図2.14中の25-21ビットのフィールドと20-16ビットのフィールド)。操作符号とレジスタファイルからの読み出しを同時に実現
- 演算の実行(EXフェーズ):演算とアドレス計算をともに必要とする命令はなく、どちらもEXフェーズ。演算装置でアドレス計算をしてもパイプラインの乱れは生じない。
- メモリアクセス(MAフェーズ):このフェーズのみでメモリアクセスが行われるため、メモリの読み書きに関する時間関係の逆転は生じない。

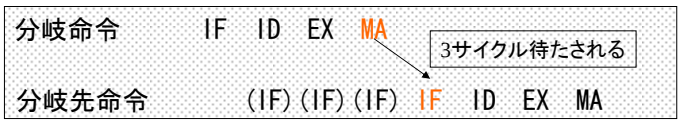
表3.5 ロード・ストア型計算機の命令実行ステップ

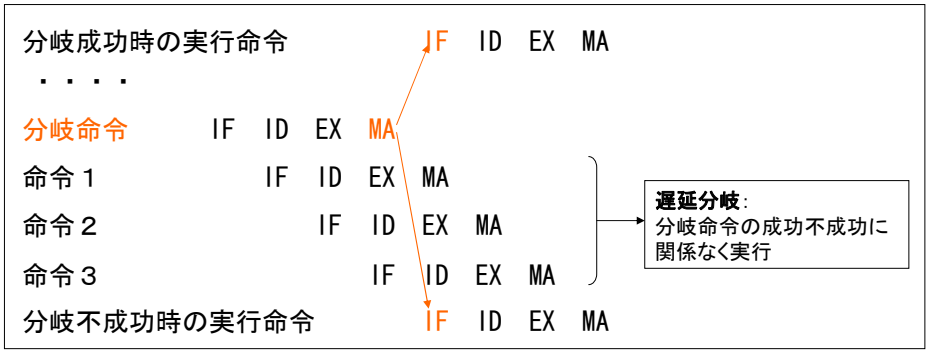
	演算命令	ロード命令	ストア命令	分岐命令
IF	命令の読み出し，PCの更新			
ID	命令の解読，レジスタの読み出し			
EX	演算の実行	アドレス計算	アドレス計算 データをMDRへ転送	アドレス計算 分岐条件の判定
MA	——	メモリから読み出し	メモリへの書き込み	分岐成功の場合 PCを更新
WB	結果をレジスタに書き込み		——	——

3.2.3命令実行のパイプライン制御

(3)ロード・ストア型計算機のパイプライン制御

- 分岐命令:分岐が成功した場合3サイクルの空きが生じる。コンパイラで対応(遅延分岐)する。

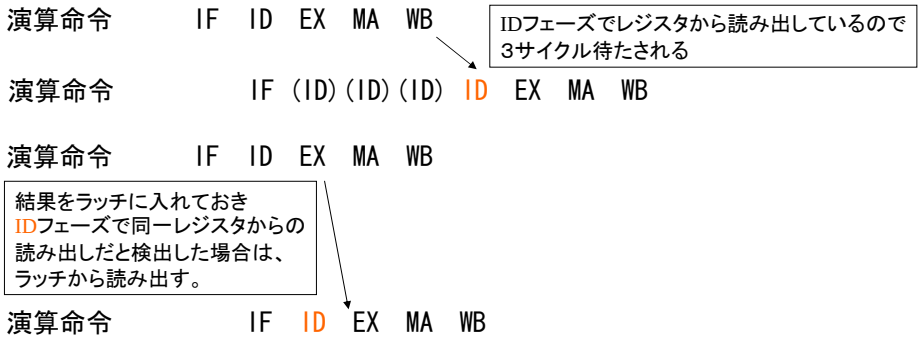




3.2.3命令実行のパイプライン制御

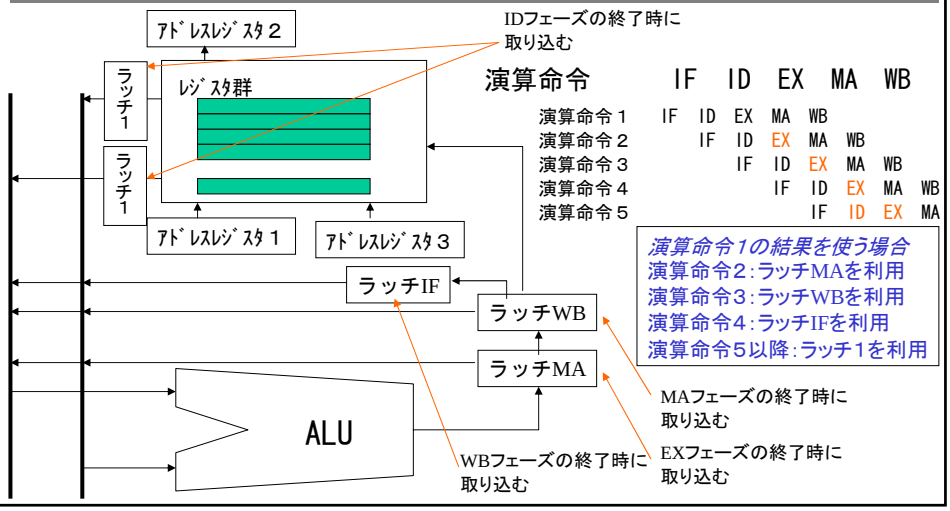
(3)ロード・ストア型計算機のパイプライン制御

- レジスタ書き込み(WBフェーズ):演算命令ではEXフェーズで結果は得られている。これを後続命令で直接利用することにより、パイプラインの乱れを回避できる。ロード命令の場合は1サイクルの空きが生じる。分岐命令同様コンパイラで対応する(遅延ロード:ロード命令でレジスタに書いたデータを次の命令でよむことを許さない)方法がとられる。



3.2.3命令実行のパイプライン制御

- RISCマシンのレジスタ構成:メモリ同様アドレス(レジスタ番号)を指定して読み書きする。読み出し回路が2つ、書き込み回路が1あり、1サイクルに同時に3つの動作が行える。



3.2.3命令実行のパイプライン制御

- RISCマシンでは、パイプライン制御が簡素化され、サイクルタイムの高速化が実現できる。
- コンパイラによる最適化技術の利用は必須。ただし、同じ機能を実現した場合、CISCマシンに比べて実行命令数は増加する。