目次

★ 7.7 前回の復習

Q1. 2 つの異なる CPU（A,B と呼ぶ）を考える．それぞれの CPU で動作する C 言語コンパイラがあり，A 用のコンパイラは A で動作する機械語プログラムを，B 用のコンパイラは B で動作する機械語プログラムを生成することができる．このとき，次の文の正誤とその理由を答えなさい．

(1) ある C 言語プログラムを上記の 2 つのコンパイラでコンパイルして得られる機械語プログラムをそれぞれの CPU で実行する場合，2 つの機械語プログラムの実効命令数は等しくなる．

(2) A,B のクロック周波数が等しいならば，ある C 言語プログラムを上記の 2 つのコンパイラでコンパイルして得られる機械語プログラムを A,B で実行したときの CPU 実行時間は等しくなる．

(3) A,B が同じ命令セットアーキテクチャに基づいていてかつ両者のクロック周波数が等しいならば，ある C 言語プログラムを上記の 2 つのコンパイラでコンパイルして得られる機械語プログラムを A,B で実行したときの CPU 実行時間は等しくなる．

(4) A で動作する 2 つの機械語プログラムの実効命令数が等しいならば，これらを A で実行したときの実行時間は等しくなる．

★8　パイプライン処理

★8.1　パイプライン処理とは

Q2. ほげお君はコインランドリー「ほげらんどり」の店長さんです．店内には，1 人分の洗濯からすすぎ，脱水，乾燥までを 40 分で行うマシンが 1 台置いてあります．このお店は 24 時間営業ですが，いつもお客さんが列をなして待っています．ということで，24 時間では 36 人のお客さんをさばける計算になります．

もっとお客さんをさばけるようにと考えたほげお君は，上のマシンを，洗濯機，すすぎ機，脱水機，乾燥機の 4 台に置き換えました．いずれも 15 分で一人分の洗濯物を処理できます．すべてのお客さんは，この 4 台を順番に使います．また，あるお客さんが例えば乾燥機を使っている間でも，他の 3 人のお客さんが洗濯機，すすぎ機，脱水機をそれぞれ使用することができます．この場合は，24 時間で何人のお客さんをさばけるでしょうか？

パイプライン処理とは，命令実行の過程を流れ作業にして，一つの命令の実行が終わるのを待たずに次の命令の実行をはじめられるようにする命令実行方式のことである．こんにちの CPU 高速化技術の要となっている．

パイプライン処理では，命令実行を複数の段階（ステージ, stage）に分割し，ある命令のある実行段階の処理と同時に別の命令の別の実行段階の処理を行う．細かい段階分けをしてステージ数を増やせば，同時に実行される命令数が多くなって命令実行のスループットを大きくできる (☆ 1)（個々の命令の実行時間が短縮されるわけではない）．このように，パイプライン処理は，個々の命令の実行時間を短縮するのではなく，命令実行のスループットを増加させることによって CPU の性能向上をはかる方法である．

説明のため，いんちき計算機 II の CPU の命令実行の手順が以下のようなステージに分けられるとしてみよう (☆ 2)．

1. 命令をフェッチする (IF: Instruction Fetch)
2. 命令をデコードする／レジスタの値を読み出す (ID: Instruction Decode)
3. 演算を実行する (EX: Execution)
4. メモリにアクセスする (MA: Memory Access)
5. レジスタへ値を書き込む (WB: Write Back)

このとき，`ADDA GR1,GR2` や `ST GR7,A` の実行手順は，次のように分解できる．

☆ 1) ただし，後述のようにパイプライン化による弊害も生ずるので，闇雲にステージ数を増やせばよいというものではない

☆ 2) これはあくまでも例であり，全ての CPU の命令実行がこのようなステージに分けられると言っているわけではない．全ての命令を同じステージ数の処理に分割できるとも限らないので，実際は話はもっと複雑になる．現代の PC 等に搭載される CPU では，ステージ数が数十に達するものもあるといわれる．

`ADDA GR1,GR2`

IF: PR が示す番地から命令を CPU 内に転送してくる
ID: それを解読して算術加算命令だとわかる．レジスタ GR1, GR2 の値を読み出せるようにする
EX: 加算を実行
MA: —
WB: 演算結果をレジスタ GR1 に書き込む

`ST GR7,A`

IF: PR が示す番地から命令を CPU 内に転送してくる
ID: それを解読してストア命令だとわかる．レジスタ GR7 の値を読み出せるようにする
EX: ストア先番地を計算
MA: その番地に GR7 の値を書き込む
WB: —

あるプログラム中でこの 2 命令が連続しているとしたら，これらを右のように重ねて実行することができる．5 ステージのそれぞれは，一度にはひとつのことだけを行っていることがわかる．

前の命令	IF	ID	EX	MA	WB			
`ADDA GR1,GR2`		IF	ID	EX	MA	WB		
`ST GR7,A`			IF	ID	EX	MA	WB	
後の命令				IF	ID	EX	MA	WB

★8.2 ハザード

上述の例のように全ての命令の命令実行ステージを均等に 5 分割できたとすると，理想的には命令実行のスループットは 5 倍になり，CPU 時間を 1/5 にすることができる．しかし，実際には様々な要因によってパイプラインがうまく流れないことがあるため，そこまでの性能向上は期待できない．パイプライン処理が乱れて命令実行が滞ることを**ハザード**という．ハザードは次の 3 種に分類できる．

ハザード: Hazard

- **構造ハザード**: 同時に行われる処理手順の組み合わせにハードウェアが対応できないことで生じるハザード．
- **データハザード**: パイプライン中で先に実行中の命令の結果に後の命令が依存する場合に生じるハザード．
- **制御ハザード**: 条件分岐命令の実行時のように，その命令の結果次第で後にどの命令を実行するかが変化するような場合に生じるハザード．

構造ハザード 上記のステージ分割では，ID ステージと WB ステージの両方でレジスタにアクセスする．したがって，これらが競合しないような工夫 (☆ 3) をしなかった場合，WB 時にレジスタに値を書き戻す命令と，ID 時にレジスタ値を読み出す命令のこれらのステージを同時に実行することができない（後の方の命令実行を遅らせるしかない）．これは，構造ハザードの例である．

☆ 3) 例えば，下図のように 1 ステージ分の処理時間の中で先に WB のレジスタアクセスを行い，その後で ID のレジスタアクセスを行うように回路を作る．

EX	MA	WB	
:	:	:	
	IF		ID

データハザード データハザードは，下図のように命令間に**依存関係**がある場合に発生する．この通りにパイプライン処理を行うと，命令 (1) の WB ステージで加算の結果が GR1 に書き戻されるより先に，命令 (2) の ID ステージで GR1 の値が読み出されてしまうので，正しい演算が行われなくなってしまう．

命令 (1) `ADDA GR1,GR2`	IF	ID	EX	MA	WB				
命令 (2) `ADDA GR3,GR1`		IF	ID	EX	MA	WB			
命令 (3) `ADDA GR4,GR5`			IF	ID	EX	MA	WB		
命令 (4) `ADDA GR6,GR7`				IF	ID	EX	MA	WB	
命令 (5) `SUBA GR2,GR2`					IF	ID	EX	MA	WB

このようなデータハザードに対応する最も簡単な方法は，命令 (2) 以降の実行を 3 ステージ分遅らせることである．しかし，このような対応ではせっかくのパイプラインを無駄にすることになる．近年では，依存関係のある命令同士がなるべく近接しないようにするソフトウェア的／ハードウェア的な工夫によってパイプライン処理の効率化がはかられている (☆ 4)．

☆ 4) コンパイラが命令の依存関係を調べ，演算結果に影響しない範囲で命令の順序を組みかえたりする．例えば上の例では，命令 (2) を (5) の後にもってくれば，演算結果も変わらずハザードも起こさない．現代の CPU の中には，実行中に命令の依存関係を検出して命令の順序を動的に組みかえることのできるものも存在する．

制御ハザード 制御ハザードは，条件分岐を含むプログラムで発生する．右の条件分岐命令 (3) はプログラムレジスタを書き換えることで分岐を実現するが，それが EX ステージで行われるとすると，それまでは次に命令 (4) が実行されるのか命令 (8) が実行されるのか確定しないので，IF をはじめられず，パイプラインを滞らせてしまうことになる (☆ 5)．

```
命令 (1)
命令 (2)
命令 (3) 条件分岐
  (条件成立時は (8) へ)
命令 (4)
命令 (5)
命令 (6)
命令 (7)
命令 (8)
```

☆ 5) ここでは EX ステージで PR が書き換わると仮定して説明しているが，PR の書き換えがもっと後のステージで行われる仕組みの CPU ならば，制御ハザードの影響はより後の命令にまで及ぶことになる．

命令 (1)	IF	ID	EX	MA	WB				
命令 (2)		IF	ID	EX	MA	WB			
命令 (3)			IF	ID	EX	MA	WB		
命令 (?)				IF	ID	EX	MA	WB	
命令 (?)					IF	ID	EX	MA	WB

制御ハザードへの対応策として，近年では，条件分岐命令を実行する際に，条件が成立して分岐するのか条件不成立で分岐しないのかを予測し，予測をもとに次の命令のフェッチをはじめてしまう（予測がはずれたらやり直す）機能（**分岐予測**と呼ばれる）をもった CPU も存在する (☆ 6)．

☆ 6) 2007 年度の高橋の「計算機アーキテクチャ」の第 9 回講義資料に少し解説があります．Wikipedia も参考になるかも．

★ 8.3 ［発展］スーパースカラ

こんにちの CPU では，命令実行ユニットを複数用意してそれぞれでパイプライン処理を行い，複数の命令を完全に同時に実行できるようにしたものも多い．このような技術を**スーパースカラ** (☆ 7) という．ただし，CPU のコア全体を複数もつ**マルチコア**との違いに注意．近年の PC 向け CPU 等では，マルチコアでかつそのそれぞれのコアが複数の命令実行ユニットをもつ，という形態になっている．

☆ 7) スーパースケーラその他いくつかの呼び方がある．

Q3. ［龍谷大学大学院理工学研究科修士課程数理情報学専攻の 2011 年度入試問題のひとつを改変したもの］ ある CPU は，命令の実行過程を授業で説明したような 5 段階に分割してパイプライン処理を行う．

(1) この CPU の 3 つのレジスタ GR1,GR2,GR3 の値の和を計算して，計算結果を GR3 に格納するために，次のアセンブリ言語プログラムに相当する機械語プログラムを作成したが，期待した実行結果が得られないという．その理由として考えられることを述べなさい．

```
        ADDA    GR1,GR2
        ADDA    GR3,GR1
```

(2) この CPU で次のようなアセンブリ言語プログラムに相当する機械語プログラムを実行すると，パイプライン処理が滞ることがあるという．その理由として考えられることを述べなさい．

```
HOGE    命令 1
        命令 2
        命令 3
        JZE     HOGE
        命令 4
        命令 5
```