目次

★1　アセンブリ言語入門その一

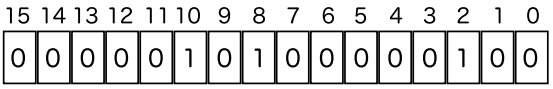
前回の「いんちき計算機」は 1 語 8bit だったが，今度は 1 語 16bit の「いんちき計算機 II」とそのアセンブリ言語「いんちきアセンブリ言語」を考えてみよう (☆ 1).

☆ 1) 「いんちきアセンブリ言語」も「いんちき計算機 II」も高橋がでっちあげた架空のものですが，独立行政法人情報処理推進機構 (IPA) の情報処理技術者試験センター (http://www.jitec.jp/) が実施している「**基本情報技術者試験**」で出題される CASLII というアセンブリ言語と，それが動く COMETII という仮想の計算機の仕様を参考にしています.

★1.1　いんちき計算機 II

仕様

- プログラム内蔵方式かつノイマン型アーキテクチャである
- 1 語 16bit
- 数値は 16bit で表現（整数のみ）．上位の bit が 2 進数の上位の桁に対応する．例えば $1284 = 2^{10} + 2^8 + 2^2$ は



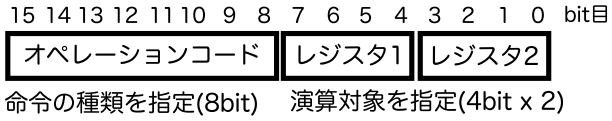
- 符号付き整数には 2 の補数表現を用いる
- 番地は 16bit で指定，1 語 1 番地
- 8 つの**汎用レジスタ**（演算結果の格納など様々な用途に使えるレジスタ）を備える (☆ 2)．それぞれ GR0, GR1,...,GR7 という名前がついている．

☆ 2) CPU 内には，汎用レジスタの他にも特別な用途に専用化されたレジスタが各種存在する．後で登場するプログラムレジスタもそのひとつ.

機械語の構成

機械語命令は 1 語または 2 語

・1 語の命令の構成 (☆ 3)



・2 語の命令の構成

☆ 3) 汎用レジスタが 8 つしかないのにレジスタ指定に 4bit 割り当てているのは，CASLII の仕様にあわせたため．将来の拡張用にとってあると考えるとよい.

★1.2　いんちきアセンブリ言語

いんちき計算機 II のための「いんちきアセンブリ言語」のプログラムの例を以下に示す (☆ 4)．アセンブリ言語プログラムの 1 行はだいたい機械語 1 命令に対応している（例外もある）．01 をならべて機械語で直接プログラムを書いたものと比べて，命令の種類（オペコード）やオペランドが記号で表されていることによってこのプログラムの動作が格段に理解しやすくなっていることがわかる

☆ 4) 命令の詳細については，「**いんちきアセンブリ言語命令集**」参照.

(☆5). また，メモリ番地を表す「ラベル」という記号をつけることができるので，番地指定が必要な命令でも番地の値をプログラマが直接扱わなくても済むようになっている．

行番号	ラベル	オペコード	オペランド
1	HOGE1	START	
2		LD	GR0, A
3		LD	GR1, B
4		ADDA	GR0, GR1
5		ST	GR0, X
6		LD	GR0, A
7		SUBA	GR0, GR1
8		ST	GR0, Y
9		RET	
10	A	DC	6
11	B	DC	10
12	X	DS	1
13	Y	DS	1
14		END	

☆5) 行番号や罫線は見やすくするためにつけたもので，アセンブリ言語の一部ではない．空白の置き方などについては，C 言語ほどの自由度はない．

Q1. 上記のプログラムを実行するとレジスタ `GR0`，`GR1` の内容がどのように変化していくか調べ，プログラム終了後に記号番地 (☆6)A,B,X,Y の内容がどのように変化するか答えよ．

上記のプログラムがメモリの `0x0108` 番地からはじまる領域に格納されたとする (☆7) と，メモリの内容は次のようになる (☆8)

番地	メモリの内容
0x0108	命令「`LD GR0, A`」を表す bit 列 (☆9)
0x0109	↑の 2 語目（A に対応した番地 `0x0115`）
0x010A	命令「`LD GR1, B`」を表す bit 列
0x010B	↑の 2 語目（B に対応した番地 `0x0116`）
0x010C	命令「`ADDA GR0, GR1`」を表す bit 列
0x010D	命令「`ST GR0, X`」を表す bit 列
0x010E	↑の 2 語目（X に対応した番地 `0x0117`）
0x010F	命令「`LD GR0, A`」を表す bit 列
0x0110	↑の 2 語目（A に対応した番地 `0x0115`）
0x0111	命令「`SUBA GR0, GR1`」を表す bit 列
0x0112	命令「`ST GR0, Y`」を表す bit 列
0x0113	↑の 2 語目（Y に対応した番地 `0x0118`）
0x0114	命令「`RET`」を表す bit 列
0x0115	整数値 6 を表す bit 列
0x0116	整数値 10 を表す bit 列
0x0117	実行前にはどんな bit 列になっているか不明
0x0118	実行前にはどんな bit 列になっているか不明

☆6) 記号番地: アセンブリ言語のプログラム中で，データが格納されているメモリ領域の番地をそのラベルで表したもの．

☆7) プログラムがメモリのどこに配置されるかは，実行の度に変わるのが普通である．また，16 bit の番地を 2 進数で表すのは大変なので，ここでは 16 進数を用いた（先頭の `0x` は「16 進数だよ」ということを表す記号）．`0x0108` は，$(264)_{10}$ であり $(0000\ 0001\ 0000\ 1000)_2$ である．

☆8) `START` や `END` はアセンブラ命令であり実際の機械語命令に対応していないのでここには存在しない．

☆9) メモリ中に “LD” といった文字列が格納されているわけではない．ここでは示していないが，前回説明したようなやり方で命令を 01 で表したものが格納されている．

★2 プログラムレジスタと分岐

★2.1 プログラムレジスタ

機械語プログラムはメモリのどこかに格納されているので，CPU がプログラムを実行する際には，機械語プログラムの命令を 1 語ずつ読み出し（**フェッチ**するという），命令を解読し，解読した命令を実行する，というステップを繰り返すことになる．CPU がメモリ中の正しい番地から命令をフェッチできるように，CPU 内には,「次に実行する命令が格納されている番地」を保持する専用のレジスタがある（**プログラムレジスタ**あるいは**プログラムカウンタ**と呼ばれる (☆ 10)）

例えば，上記のプログラムが上記のようにメモリに格納されているとして，丁度 3 行目の LD 命令の実行が終わったところとすると，プログラムレジスタの内容は `0x010C`(☆ 11) になっている．このとき，CPU が次の命令を実行する過程は次のようになる．

1. 命令をフェッチして解読し，PR をインクリメント（1 を加える）する
   (a) CPU が PR の値を読み出し，そこに記された番地の内容を CPU 内に転送（フェッチ ）
   (b) フェッチした命令を解読．1 語の加算命令 (`ADDA GR0, GR1`) だとわかる
   (c) PR の値に 1 を加える (PR の値は `0x010D` になる)(☆ 12)
2. その命令を実行する (`ADDA GR0, GR1`)
3. 次の命令へ（次は `0x010D` からフェッチされることになる）

ここでは示していないが，2 語命令の場合は「フェッチして PR をインクリメント」という動作を 2 回繰り返すことになる．

☆ 10) Program Register を略して PR と表記したり，Program Counter を略して PC と表記することが多い（Personal Computer ぢゃない）．

☆ 11) bit 列で表せば 0000 0001 0000 1101.

☆ 12) 1 語 1 番地で番地のついた計算機を考えているから PR には 1 を加えているが，例えばバイト単位で番地のついた 1 語 32bit の計算機なら，4 を加えることになる

★2.2 無条件分岐

上述のように，CPU は命令をフェッチする度にプログラムレジスタ (PR) の値を自動的にインクリメントしていくので，プログラムはメモリ格納番地の小さい方の命令から順に実行されていく．しかし，PR の値を書きかえる命令があれば，その次の命令をどこからフェッチさせるかプログラムの側でコントロールすることができる．このようにプログラムの処理の流れを変えることを**分岐**という．ここでは，処理の流れを常に同じ場所に変更するような分岐である**無条件分岐**を考える．条件に応じて分岐先を変える（分岐したりしなかったり）する**条件分岐**については後ほど．

いんちきアセンブリ言語では，`JUMP` 命令で無条件分岐させることができる．例えば次のようなプログラムを考えてみよう．

	ラベル	オペコード	オペランド
1	LOOP	ADDA	GR0, GR1
2		SUBA	GR0, GR1
3		JUMP	LOOP
4		ADDA	GR1, GR2

命令実行の流れは次のようになる．この場合，無限ループになり，4 行目の命令は決して実行されない．

1. 1 行目の命令をフェッチして実行．PR は次の命令の格納番地を指す
2. 2 行目の命令をフェッチして実行．PR は次の命令の格納番地を指す
3. 3 行目の命令は JUMP 命令なので，実行すると PR がオペランドで指定された値に書き換えられる．この場合オペランドは LOOP というラベルであるから，PR には LOOP というラベルがついた命令（すなわち 1 行目の命令）の格納番地がセットされる
4. PR は 1 行目の命令の格納番地を指しているので，それをフェッチして実行．PR は次の命令の格納番地を指す
5. 2 行目の命令をフェッチして実行．PR は次の命令の格納番地を指す

: