アーキテクチャ　第５回　追加メモ　アドレシング・命令セット・性能

それぞれの命令の動作、書き方、それがメモリ上で 0/1 でどう書かれるか　⇒　COMET-II 命令表を見る

第1回目に配布した資料 「別紙1　アセンブラ言語の仕様」 がキーなので、これの見方を理解するとよい。

1．1節はとにかく読んで、ふ～んと思って欲しい。後で必要になる。
1．2節が1つ1つの命令の説明なので、ざっとみておくこと。
真ん中の書き方の欄が上下に分かれているものは、2とおり（レジスタ間の命令と、レジスターメモリ間の命令）がある。
実効アドレスや adr の解釈は、授業で説明しました。 命令の中の adr フィールドに書かれた値に、アドレッシングモード（COMET-II は実質インデックスモードしかない）による修飾を加えた値が「実効アドレス」となり、メモリアクセスに使われる。
命令の説明の欄は、たとえば LD 命令であれば、r1←(r2) であれば r2 の内容を r1 に代入（コピー）する、と読む。

2節はほとんどとばしてよい。例外は2．7節で、r とか r1, r2, x と書いていたところに GR0～GR7 を書けとか、adr と書いていたところには 10 進又は 16 進の数を書けと書いてある。16 進数を区別するために、この授業では 0x3F のように先頭に 0x を付けることにする。

9ページ以降の「参考資料」には、機械命令のフォーマット（形式）が書かれている。
命令には16ビット（＝1語）のものと、32ビット（＝2語）のものがある。それは中央にある「命令語長」の欄で区別される。
1語の命令の場合は第1語のみで完結する。2語の命令の場合は第1語と第2語が繋がった形になる。 第2語は表を見て分かるように adr しか入らない。
第1語は上位の8ビット（ビット15～8）がOP部、その次の4ビット（ビット7～4）がレジスタオペランド r または r1 を指定するフィールド、その次の4ビット（ビット3～0）がインデックスアドレシングで修飾に使うインデックスレジスタとして、どの（汎用）レジスタを使うか（GR1～GR7）を指定する。但し、GR0を指定することは出来ず、x として 0 を指定すると「インデックスアドレシングをしない」つまり「直接アドレシングをする」という意味になる。
OP部は更に4ビットずつに分かれて解釈される（表にある通り）が、あまり考え込む必要はない。パターンだと思えばよい。

この程度の理解を前提にして、話を進める。

「何を覚えたらいいのですか」

繰り返しになるが、「覚える」よりは、どう考えるかを「理解して」欲しい。
覚えたことは、試験が終わると忘れてしまう。 無味乾燥な、つながりのない事実を覚えても、多分役には立たない。
事実の繋がりを、ストーリーとして理解しておくと、その中の1カ所をつついただけで残りの部分がぞろぞろと繋がって出てくるはずだ。それが一部欠けていても、特にその中の具体的な数や値が欠けていても、それは調べれば済むことだし、調べるための繋がり＝きっかけが分かっていれば、調べることができるはずだ。
特にこの授業では、この後に書いておく「設計」の考え方を前提として、「昔の人がどう考えたので、今こうなっている」というストーリーを理解して欲しい。 今後はそのストーリーに関する知識を応用して、新しいものを考えたり、より上手にコンピュータを使いこなしたりできるはずだ。

「機械命令（機械語・アセンブラ言語）を使ってプログラミングを試してみたいのですが」

授業ホームページ日程一覧表の「2012 年版予習問題」の第6回分の後半に、やり方が書いてあるので参考にして下さい。

「コンピュータ（や CPU）をどう設計するか」を考える

前回の質問で目立ったのが、「なぜ、１－オペランドやら２－オペランドを区別するのか、３－オペランドなら全部指定できるのだから、それがあればいいではないか」 といったものだった。 これに限らず、「アーキテクチャ」の授業では≪選択肢≫の議論が、あちこちに出て来る。 なぜ「選ぶ」のか？ どういう基準で「選ぶ」のか？

「設計」とは？

高校までのさまざまな勉強で学んできたことは、「ここはこうなっている」とか「こうすれば計算できる」とかのような話だったと思う。（理系の）大学で扱う疑問は、たとえば

- 宇宙はどうなっているのか？ 物質はどう構成されているのか？ 生物はどうやって生きているのか？ といった自然科学系の疑問と、
- 自動車を・橋を・飛行機をどうやって作るのか？ プラスチックをどうやって安く大量に作るのか？ コンピュータをどうやって安く・早く・小さく作るのか？ といった工学系（物作り）の疑問と

があると思う。 もちろんその他にも、こういうものを解き明かして／作れるようになってよいのか？（倫理的な問題）、とか、ユニークなアイデアをどう尊重するのが良いのか？（社会制度・法律などの問題）、なども考えなければならないが。
このアーキテクチャの授業は、後者（コンピュータをどうやって安く・早く・小さく作るのか？）に属するだろう。「設計」と呼ぶこともある。

「設計」のプロセスは、自分の考えている目標（コンピュータを安く・早く・小さく作る）を実現するために、どのようにしたらよいかを決めてゆくことなのだが、実態は、実現可能なやり方（選択肢）の中から良いものを選んで組合せてゆくというプロセスになる。 選択には、個々の要素を選択肢から選ぶことも大事だし、要素を組み合わせた時にうまくゆくことも大事だろう。

選択する時に大事なことは、どういう基準で選ぶか、つまり「何がよくなるからこれを選ぶ」というその基準である。だが、その基準は絶対ではない。たとえば乗用車を設計する時に、スピードレース用の車を作りたいのであれば燃費も居住性も騒音も無視して設計するだろう。 他方、街中で使うためであれば、燃費や居住性が大切かも知れない。「これさえ高めればOK」というような絶対の基準は、存在しないのが普通である。
目的が決まったとしても、１つの基準さえ最高にすればよいわけではない。 街中で使う乗用車にしても、「居住性が良ければスピードが出なくてもいい」とは言えないのである。 高速道路を楽に走れる程度の加速性能は欲しいし、高速道路で追い越しをする時に十分な程度の高速安定性が欲しい。

更にやっかいなのは、２つの基準を高めようとするときに、それらが両立しない、背反する、つまり一方をよくすると他方が悪くなるという要素が、結構多いことである。 居住性をよくするために車体を大きくすると、車体重量が重くなるので、加速性能は悪くなるだろう。 レーシングカーは加速性・高速性を追求するために、居住性を犠牲にする。 両立しないからである。 設計の過程では、このような二律背反がたくさん出てくる。 それらを上手に妥協して（うまい妥協点を見つけて）、全体として求めるものを作り上げるのが、上手な設計になる。 但し、時代によって技術が新しく作られるので、たとえば軽い材料が得られるようになれば大きくても軽くて高速なものが作れるようになるかも知れない。

つまり、設計とは、二律背反するようなさまざまな選択肢から、その時点で最良の組合せを選択する過程である。

オペランドの数の選択

オペランドの数の選択も、CPU の設計の一部になる。 今 CPU を１つ作らなければならないとして、どのような命令の形に

するか、つまりOP部が何ビット、オペランドの数が幾つ、それぞれのオペランドはメインメモリから持ってくるのか汎用レジスタか、アドレッシングモードなどの修飾はどれだけ指定できるようにするか、などなど、決めて下さいと言われたとしよう。 何を考えて決めればいいのだろうか？

さまざまな選択の項目があるわけだが、全体として満たしたい基準は単純でないのでやっかいである。 まず、命令実行が十分に速いことが求められる。他方でプログラム（命令が並んでいるもの）はメインメモリに置くので、あまり大きくなるとメインメモリが足りなくなったりして困る。1つ1つの命令があまり長く（ビット数が多くく）ならない方が良い。 また、プログラムのしやすさ・書きやすさも求められる。機械命令でプログラム作成することはほとんど無いので要らないように考えるが、実はコンパイラでコンパイルするときに、無駄のない効率の良いプログラムが生成できるかどうかは結構重要である。
1つ1つの命令の長さを考えてみよう。 3－オペランド方式は3つ指定しなければならないのだから、3つを指定するためのフィールドが命令中に必要になる。 オペランドの指定は、メインメモリ上のオペランドをアドレスで指定する場合と、汎用レジスタ上のオペランドを汎用レジスタの番号で指定する場合がある。 メモリ上のオペランドを指定するには、メモリのアドレスを書かなければならない。メモリのアドレスの長さは、システムによって異なるが、COMET-II では16ビットであり、我々が使うパソコン（Intel x86 系の CPU）では32ビット（32ビット全部が直接メモリアドレスになるのではなくて「仮想記憶」の「ページテーブル」を選択する。2年生のOSの授業で学ぶ）であるから、3つのオペランド全てを32ビット（＝4バイト）で指定するとオペランド部分だけで3×4バイト必要になり、OP 部と合せると更に必要になる。
COMET-II のように、2－オペランドにして、しかもその一方は汎用レジスタの番号（0～7）を指定することにすると、命令のビット長は大分短くなる。 まず、オペランドが2つとも汎用レジスタであれば、それぞれの番号を4ビットで指定するので、8ビットあれば済む。 したがって COMET-II の短いタイプの命令は、OP 部の8ビットと合せても16ビットで済む。 一方のオペランドがメインメモリのアドレスを指定し、もう一方は汎用レジスタの番号を指定する、長いタイプの命令では、OP 部が8ビット、メインメモリ上のオペランド指定がアドレス部分が16ビットとインデックスレジスタの番号を指定する4ビットで20ビット、もう一方の汎用レジスタのオペランドを指定するのに4ビット、全部で32ビットになる。

別の尺度として、プログラムが無駄がなく書けるかどうかを考えてみる。 3－オペランド方式で且つ3つのオペランドがメインメモリでも汎用レジスタでも選択できるようなシステムであれば、コンパイラから見た選択の自由度が高いので、もっとも効率の良いプログラムが生成できる。2－オペランドになると、必ず一方のオペランドを上書きすることになるから、もし後で必要になるならば事前に退避コピーを作っておかなければならない。また COMET-II のように一方のオペランドが汎用レジスタに制限されていると、もし結果をメインメモリ上の変数に格納したければその後に追加でストア命令を置かなければならない。つまり余分な実行時間を食うことになり、損になる。 およその話、演算の入力オペランドの読出し先、出力オペランドの書込み先が自由に選べるなら、もっとも無駄のない（つまり高速になる）データの動きを作ることができるが、オペランドに制約条件が付くと（2－オペランドのように入力側の1つと出力側が同じでなければならない、プラス COMET-II だとそれが汎用レジスタでなければならない）、つじつまを合わせるためには余分な命令が必要になり、遅くなる。

設計という目で見れば、オペランド数を増やすと、コンパイラに対する自由度は増えるが、命令を置くメモリのスペースが増える。オペランド数を減らせば、スペースは減るが、コンパイルしたプログラムに無駄ができ遅くなる。 二律背反になっている。 落としどころは、作りたいシステムの目的（レーシングカーなのかタウンカーなのか）と、その時の技術やコストで決まる。メインメモリの量は、昔はコストが高かったのでわずかしか持てなかったが、最近は安価になって大量に持てるようになった。だから置きたければ長い命令でも置ける。但し CPU 内での扱いが未だわずらわしいので、あまり長い命令は具合が悪い。結局、メモリ＋レジスタの2－オペランド方式が広く使われていたりする。 また、RISC 設計をしたプロセッサでは、3－オペランドで汎用レジスタ3つの指定をするものもある（ARM プロセッサはそうらしい）