アセンブリ言語と機械語

樋口さぶろお

龍谷大学理工学部数理情報学科

情報処理の基礎　L11(2014-12-17 Wed)

今日の目標

- 表を参照して, 特定のことが起きる機械語プログラムを書ける
- 機械語プログラムがビットパターンに符号化できることを説明できる
- 「機械語命令に従って動く CPU の気持ちになれる」

http://hig3.net

L10-S1

Quiz(機械語)

この状況で, アドレス 1000 から 100C の機械語命令が順に実行されると, レジスタとメモリの内容はどうなる?

CPU
PC: 1000
レジスタ
A:00001F01
B:00002A03

メモリ
アドレス:セルの中身
1000: 読込 | A | 1204 |
1004: 定数読込 | B | 01208 |
1008: 演算 (加算) | A | B |
100C: 書出 | A | 1204 |
1010:
⋮
1200: 00001204
1204: 00001010
1208: 0000000A
120C: 0000000B
⋮

Quiz 略解+コメント:機械語

略解

A: 00002218

B: 00001208

1200: 00001204

1204: 00002218

1208: 0000000A

121C: 0000000B

L10-S3

Quiz(機械語)

すべて 16 進法.
ここから実行が始まり，0114 の命令が実行されようとするときの CPU とメモリの状況は? もしレジスタ A と B の中身が逆だったら?

CPU
PC: 0100
レジスタ
A:00001200
B:00001204

メモリ
アドレス:セルの中身
0100: 演算 (減算) A B
0104: 条件分岐 0110
0108: 書出 B 1200
010C: 分岐 0114
0110: 書出 A 1200
0114:
⋮
1200: 00001010
1204: 00001200
⋮

Quiz 略解+コメント:機械語

略解

条件分岐では実際に分岐する.

PC: 0114

A: FFFFFFFC (-4 の, ビット長 32 の, 2 の補数表現)

B: 00001204

1200: FFFFFFFC

1204: 00001200

A,B が逆だったら, 条件分岐で分岐しない.

A: 00000004

B: 00001204

1200: 00001204

1204: 00001200

ここまで来たよ

1. 略解:CPU の仕組み

2. アセンブリ言語と機械語
   - 機械語/アセンブリ言語プログラミング
   - アセンブリ言語と機械語
   - もっと機械語プログラミング
   - キャッシュ

命令一覧

	20b 定数,	16b アドレス, A or B のレジスタ指定
0	読込	メモリのアドレスの値をレジスタにコピー
1	定数読込	定数をレジスタにコピー
2	書出	レジスタの値をメモリのアドレスにコピー
3	演算 (加算) A B	レジスタ A と B に対して ALU を使って加算 A + B を行い, 結果を A におく
4	演算 (減算) A B	A − B. 結果が負なら条件フラグを 1 に.
5	演算 (乗算) A B	A × B.
6	演算 (除算) A B	A / B.
7	分岐	PC の値をに変更する
8	条件分岐	条件フラグがセットされているときだけ PC の値をに変更する. 条件フラグを 0 に

Quiz(機械語)

メモリ

⋮

1200: 整数 x

1204: 整数 y

1208: 整数 z

120C: ??

1210: ??

1214: ??

1218: ??

121C: ??

⋮

この状況から実行を始めて, 整数 $r = (x + 2 \times y) \times z$ をアドレス 120C に書き出す機械語プログラムを書こう.

Quiz(機械語)

メモリ

⋮

1200: 整数 x

1204: 整数 y

1208: 整数 z

120C: 整数 w

1210: ??

1214: ??

1218: ??

121C: ??

⋮

この状況から実行を始めて, 整数 $r = (x + y) \times (z - w)$ をアドレス 1210 に書き出す機械語プログラムを書こう.

ここまで来たよ

1. 略解:CPU の仕組み

2. アセンブリ言語と機械語
   - 機械語/アセンブリ言語プログラミング
   - アセンブリ言語と機械語
   - もっと機械語プログラミング
   - キャッシュ

これまで使ってたのはなんちゃってアセンブリ言語

なんちゃってアセンブリ言語

書出 B 1204
定数読込 A 00001

アセンブリ言語 (のひとつ) では, ニーモニックで次の様に書く

```
ST B, 1204
LDI A, 00001
```

LD (オペレータ)　A (オペランド 1), 1200 (オペランド 2)

オペレータ 作用するもの, 演算子, 命令, 動詞
オペランド 作用されるもの, 引数, 目的語

アセンブリ言語は［　　］の機種ごとにある. 移植性がない.
アセンブリ言語の例 CASL2 情報処理技術者試験での出題に使われる理論上のアセンブリ言語

オペコード	ニーモニック	この授業での表記	元の英単語
0	LD A, 1204	読込	load
1	LDI A, 0	定数読込	load immediate
2	ST B, 1200	書出	store
3	ADD A, B	演算 (加算) A B	add
4	SUB A, B	演算 (減算) A B	subtract
5	MUL A, B	演算 (乗算) A B	multiply
6	DIV A, B	演算 (除算) A B	divide
7	JAL 112	分岐	jump all
8	JLT 112	条件分岐	jump less than

高級言語 ↔ 低級言語

C 言語 $\overset{\text{コンパイル}}{\rightarrow}$ アセンブリ言語 $\overset{\text{アセンブル}}{\rightarrow}$ 機械語

コンパイラ コンパイル (compile) するプログラム 例:cc
アセンブラ アセンブル (assemble) するプログラム 例:as

Linux の cc は, コンパイルした後 as を呼び出す.
コンパイラってどんな仕組み? C 言語の入力に対してアセンブリ言語を出力する

記号処理 (3 前)

高級言語	低級言語
人にやさしい	CPU にやさしい
抽象的	具体的
移植性	移植性
機械語命令と対応しない	1 行=1 機械語命令
プログラム短い	プログラム長い

命令の符号化

定数読込 [B] [00001] ・・・アセンブリ言語

→ 1 1 00001 0

4bit 4bit 20bit 32bit になるように穴埋め

→11 00 00 10 ・・・機械語・マシン語

- オペレータは読込,…, 条件分岐 の 9 個あるので通し番号で表現 [ ]. 4bit
- レジスタ 2 種類. 1bit ですむけど 4bit
- アドレス そのまま 16bit
- 定数 そのまま 20bit
- あまったところは 0 でうめる.

実際には命令もレジスタももっと多くある

例

- 定数読込 [B][00001]=1 1 00001 0 → 12 00 00 10
- 演算 (加算)[A][B]=3 0 1 00000 → 30 10 00 00
- 書出 [A][1200]= 2 0 1200 00 → 20 12 00 00

コンピュータの中ではなんでもビットパターン

ここまで来たよ

1. 略解:CPU の仕組み

2. アセンブリ言語と機械語
   - 機械語/アセンブリ言語プログラミング
   - アセンブリ言語と機械語
   - もっと機械語プログラミング
   - キャッシュ

人間コンパイラ体験

次の C プログラム (片) に相当する機械語を求めよう

```
int i, sum;
sum = 0;
for(i=2; i<=9; i=i+1){
    sum = sum + i;
}
```

おおざっぱにいえば

制御構造 if, while, for は条件分岐で実現

(プログラム中で値の変わる) 変数 メモリ中の自分で決めたアドレスに置き, レジスタとの間で読込, 書出する

定数=(プログラム中で値の変わらない変数) プログラム中にオペランドとして書いて, レジスタに定数読込する

レジスタは臨時の置き場. ここに特定の変数を置くのではない.

```
int i, sum;
sum = 0;
for(i=2; i<=9; i=i+1){
    sum = sum + i;
}
```

これは正しいけど愚直なプログラム. 現在の値が $A = 12$ ってわかってるのに再度 代入 $A = 12$ をするような (書出すぐ読込みたいなこと) ことしてる. もっと短く書き直すと?

これを機械語 (ビットパターン) に変換することを真剣に想像してみよう

```
1000: 定数読込[A][00000]
1004: 書出[A][2004]
1008: 定数読込[A][00002]
100C: 書出[A][2000]
1010: 読込[A][2000]
1014: 定数読込[B][00009]
1018: 演算(減算)[B][A]
101C: 条件分岐[103C]
1020: 読込[B][2004]
1024: 演算(加算)[B][A]
1028: 書出[B][2004]
102C: 定数読込[B][00001]
1030: 演算(加算)[A][B]
1034: 書出[A][2000]
1038: 分岐[1010]
103C:
...
2000:????????  /*i*/
2004:????????  /*sum*/
```

ここまで来たよ

1. 略解:CPU の仕組み

2. アセンブリ言語と機械語
   - 機械語/アセンブリ言語プログラミング
   - アセンブリ言語と機械語
   - もっと機械語プログラミング
   - キャッシュ

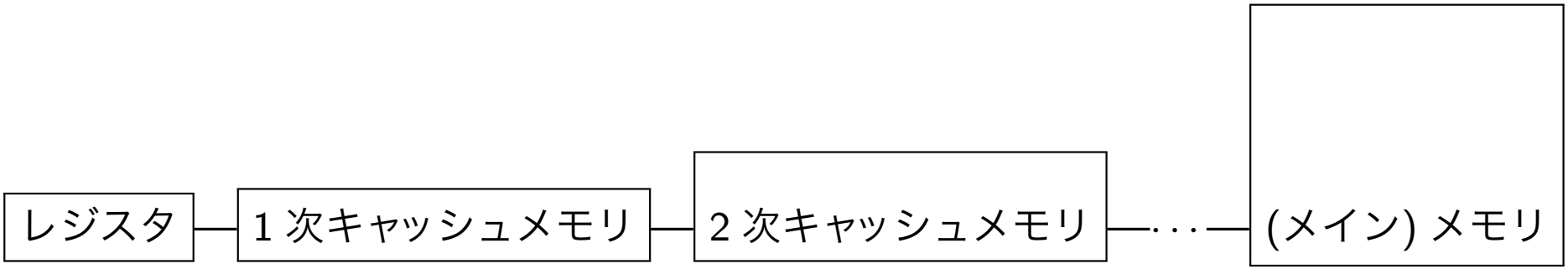
キャッシュによる高速化

(メイン) メモリとレジスタの間では頻繁にデータが転送される．そのたびにレイテンシの分だけ時間をロスする．

対策 1 もっと高い (メイン) メモリを買う
対策 2 CPU との間の読み書きをキャッシュメモリに対して行い，キャッシュメモリとメインメモリの間はまとめて読み書きする

例え話
帳簿庫と事務机の間では頻繁に帳簿が転送される．そのたびに，移動する分だけ時間をロスする．

対策 1 もっと近い帳簿庫に参考書を置くように交渉する
対策 2 事務机に近い  まとめて写す．



暗黙の仮定

プログラムにしろ, データにしろ, メインメモリ上で近いアドレスに記憶した情報を頻繁に使う.

	1 次キャッシュ	2 次キャッシュ	メインメモリ
ハードウェア	SRAM	SRAM	DRAM
記憶できる情報量	小 (MB)	中	大 (GB)
レイテンシ	小 (速)	中	大 (遅)
情報量あたりの価格	高	中	安

連絡

- 次回の予習問題は 2015-01-06 火 23:55 まで
- **レポート問題あり** 2015-01-15 木 ごろまで. RaMMoodle から個人別問題取得.
- 2015-01-07 水 3,2015-01-14 水 3 ふつうの授業
- 2015-01-14 水 5 たぶんプチテストリベンジ 希望者のみ参加. プチテストと同じ出題計画. これと前回のプチテストのうち, 点数の高いほうを 30 ピーナッツ分とします. プチテストで 70% 以上を得た人は, 無視してファイナルトライアルに集中することをおすすめします.
- 2015-01-21 水 たぶん補講
- 2015-01-28 水 たぶんファイナルトライアル (40 ピーナッツ)
- 配布資料は 1-503 向かいの引出, `http://hig3.net` で再配布.
- Quiz の略解は `http://hig3.net` で配布しています.
- 予習問題, 成績や略解は `http://hig3.net` → RaMMoodle から
- 非参照非相談テストの答案や成績や略解は `http://hig3.net` → RaMMoodle から