情報工学実験 IV
第 1 回： 16 ビット RISC マシンの実装

柴田裕一郎

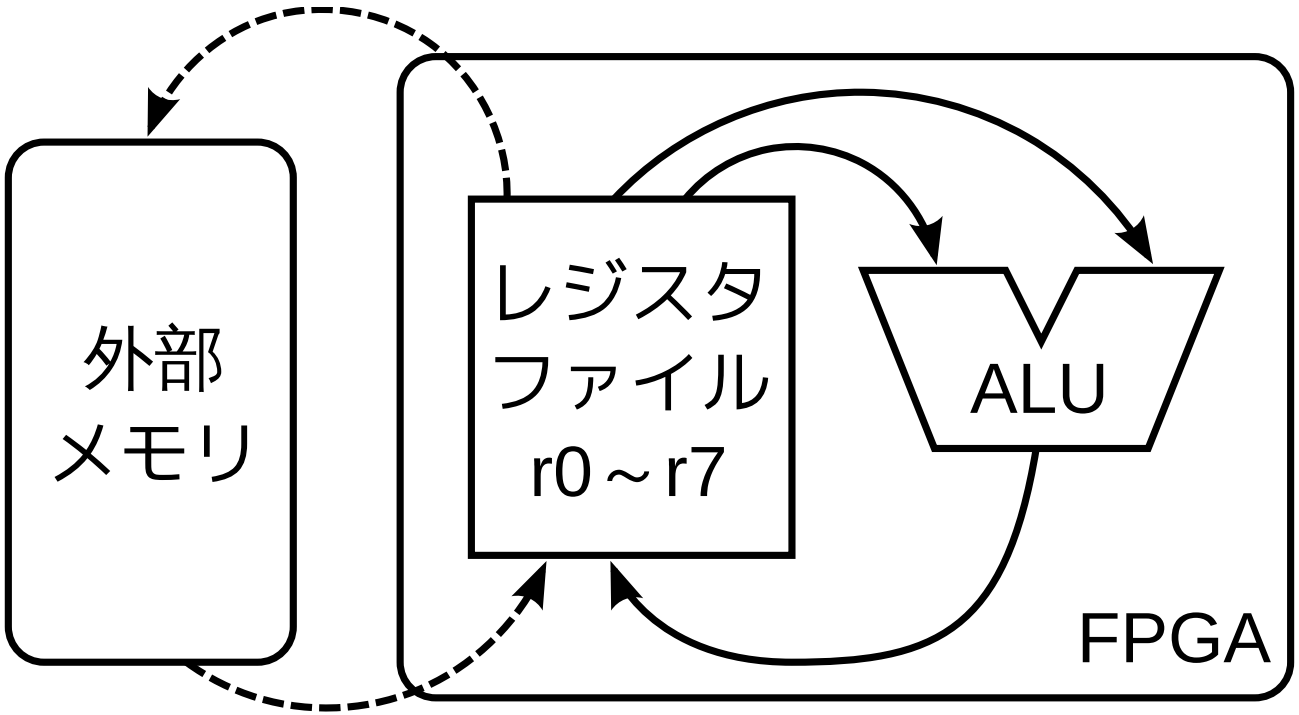
shibata@cis.nagasaki-u.ac.jp
情報工学コース

2018 年 11 月 27 日

RISC（Reduced Instruction Set Computer）

- 単一固定の命令長を持つ
- レジスタ・レジスタアーキテクチャである
  - レジスタとレジスタの間で演算を行う
- RISC という名前そのものは「命令の種類を減らした」ということを意味
  - 命令の機能を均一化し高機能命令を削除
  - 近年ではマルチメディア演算用命令などが追加されている
  - 必ずしも「命令の種類が少ないこと」は RISC の本質ではない

RISC16 の大雑把な構造



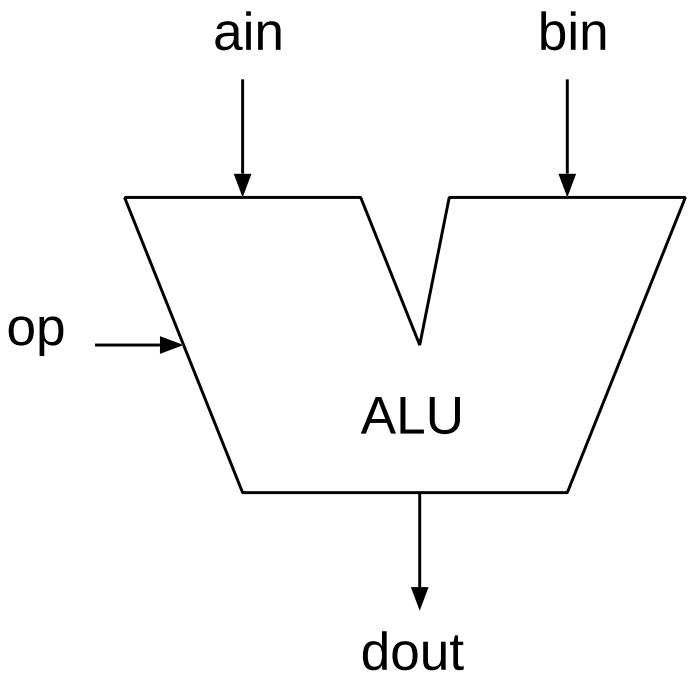
メモリ

- SRAM（Static RAM）
  容量は小さいが高速．インバータ（NOT 素子）でループをつくって値を覚えさせる．アクセスの手順は簡単で使いやすい．プロセッサのキャッシュメモリなどに使われる．
- DRAM（Dynamic RAM）
  容量が大きいがやや低速．半導体で作ったコンデンサの電荷に値を覚えさせる．アクセスの手順がやや複雑．PC などのメインメモリ（主記憶）などに使われる．

今回は FPGA 外部の SRAM を使用： 論理合成の対象ではない

ALU のインタフェース

- 加減算の他に論理演算などを行う組み合せ回路
- 通常は乗算や除算などのマルチサイクルの演算は扱わない
- データ入力： ain，bin （各 16 ビット）
- データ出力： dout （16 ビット）
- 演算指定の入力： op （4 ビット）



16 ビット ALU でサポートする演算

op の値	動作
0000	ain をスルーしてそのまま出力（A）
0001	bin をスルーしてそのまま出力（B）
0010	bin の否定（$\overline{B}$）
0011	排他的論理和（$A \oplus B$）
0100	加算（$A + B$）
0101	減算（$A - B$）
0110	bin を左へ 8 ビットシフト（$B << 8$）
1000	bin を左へ 1 ビットシフト（$B << 1$）
1001	bin を右へ 1 ビットシフト（$B >> 1$）
1010	論理積（$A \wedge B$）
1011	論理和（$A \vee B$）

- シフトでは余ったビットには ‘0’ をつめる（論理シフト）

ALU の記述例

```
1  `define ALU_THROUGH_AIN 4'b0000
2  `define ALU_THROUGH_BIN 4'b0001
3   ...
4  `define ALU_SHIFT_LEFT  4'b1000
5   ...
6
7  module alu16
8    (
9     input wire [15:0]   ain, bin,
10    input wire [3:0]    op,
11    output logic [15:0] dout
12    );
13
14    always_comb begin
15       case (op)
16         `ALU_THROUGH_AIN: dout <= ain;
17         `ALU_THROUGH_BIN: dout <= bin;
18          ...
19         `ALU_SHIFT_LEFT:  dout <= bin << 1;
20          ...
21        default:          dout <= 16'bx;
22       endcase
23    end
24 endmodule
```

Don't care

- 組み合わせ回路記述での不定値 "x" の代入

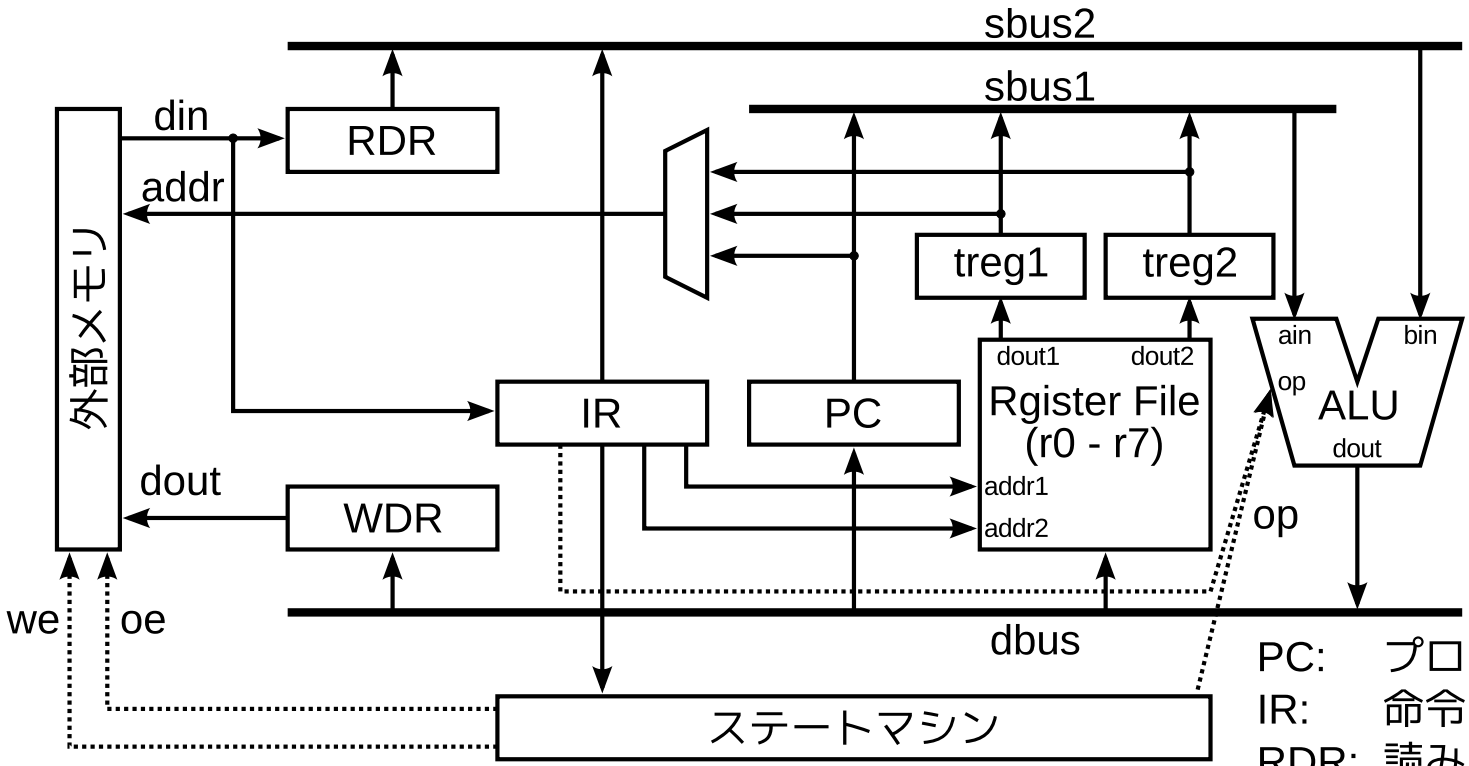
```
default:                dout <= 16'bx;
```

- don't care を意味する
- 論理合成の際に，なるべく回路が簡単になるように適当な値が選ばれて代入される
- シミュレーションでは不定値として処理される

RISC16 の仕様設計

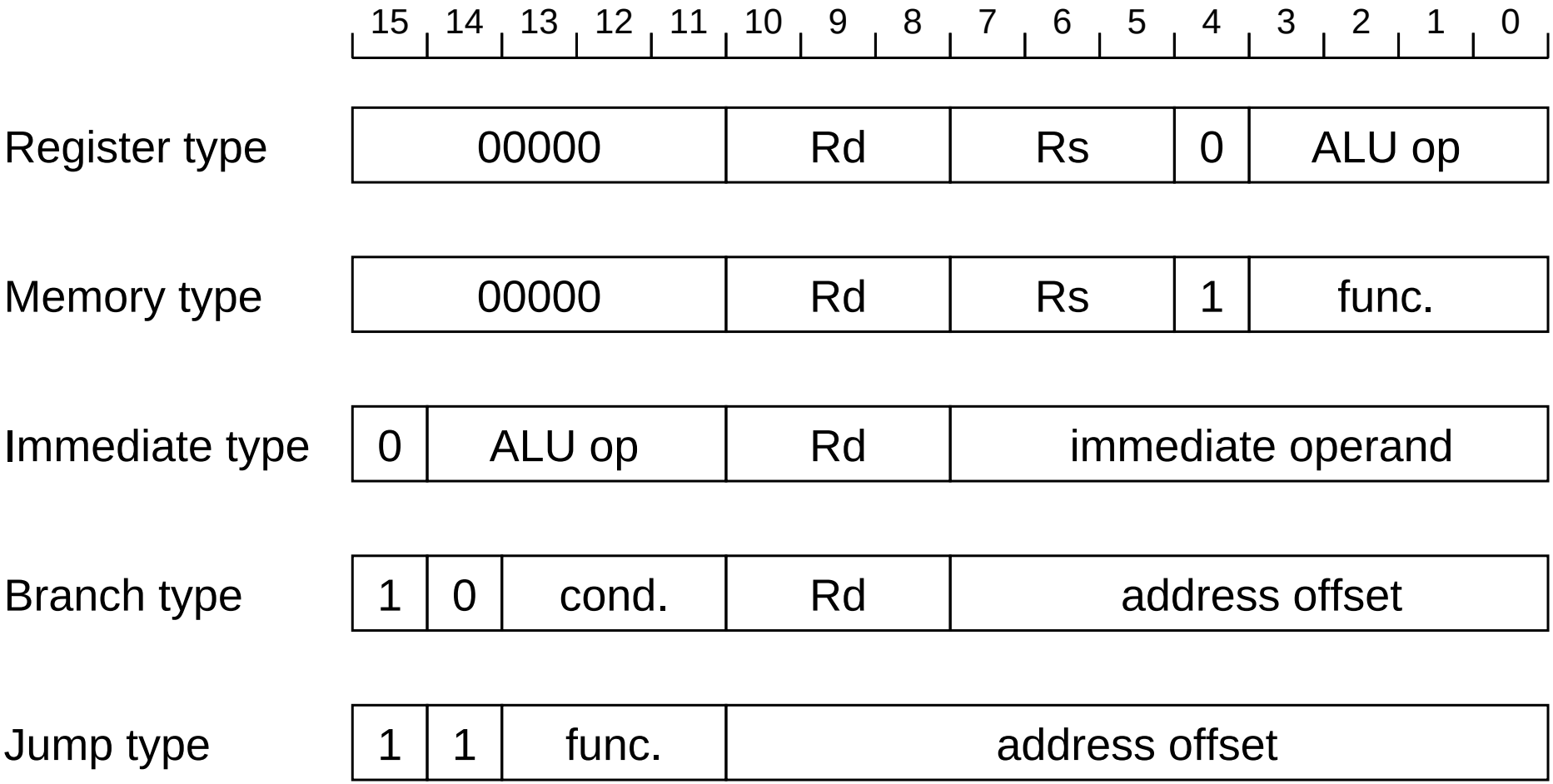
- レジスタ・レジスタアーキテクチャ
- 16 ビット × 8 本の汎用レジスタファイル
  汎用レジスタ … 使い方が限定されていない
- レジスタ間接アドレッシング
  - レジスタに入っている値を番地としてアクセス
  - （例）r1 に格納されている番地に r0 の内容をストア
- アドレス空間は 64K バイト
  アドレスバスは 10 ビットから 16 ビットに拡張
- 分岐先アドレスは PC 相対指定

RISC16 の構成



PC: プログラムカウンタ
IR: 命令レジスタ
RDR: 読み出し用データレジスタ
WDR: 書き込み用データレジスタ

命令フィールディング


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Register type
00000
Rd
Rs
0
ALU op
Memory type
00000
Rd
Rs
1
func.
Immediate type
0
ALU op
Rd
immediate operand
Branch type
1
0
cond.
Rd
address offset
Jump type
1
1
func.
address offset

レジスタ命令タイプの主な命令

命令コード	ニーモニック	動作
0000000000000000	NOP	有効な操作は行われない （r0 を r0 に移動）
00000dddsss00001	MV *d*, *s*	レジスタ s の内容を d に格納
00000dddsss00010	NOT *d*, *s*	レジスタ s の反転を d に格納
00000dddsss00011	XOR *d*, *s*	レジスタ d と s の排他的論理和を d に格納
00000dddsss00100	ADD *d*, *s*	レジスタ d と s の和を d に格納
00000dddsss00101	SUB *d*, *s*	レジスタ d から s を引いて d に格納
00000dddsss01000	SL *d*, *s*	レジスタ s を 1 ビット左シフトして d に格納
00000dddsss01001	SR *d*, *s*	レジスタ s を 1 ビット右シフトして d に格納
00000dddsss01010	AND *d*, *s*	レジスタ d と s の論理積を d に格納
00000dddsss01011	OR *d*, *s*	レジスタ d と s の論理和を d に格納

メモリおよびイミディエイト命令タイプの主な命令

命令コード	ニーモニック	動作
00000dddsss10000	ST d, (s)	レジスタ d の内容を s で示す番地に格納
00000dddsss10001	LD d, (s)	レジスタ s で示す番地の内容を d に格納

命令コード	ニーモニック	動作
00100dddxxxxxxxx	ADDI d, #X	レジスタ d と値 X（符号拡張）の和を d に
01010dddxxxxxxxx	ANDI d, #X	レジスタ d と値 X の論理積を d に格納
01011dddxxxxxxxx	ORI d, #X	レジスタ d と値 X の論理和を d に格納
00001dddxxxxxxxx	LLI d, #X	値 X をレジスタ d に格納
00110dddxxxxxxxx	LUI d, #X	値 X をレジスタ d の上位 8 ビットに格納

分岐（ブランチ）およびジャンプ命令タイプの主な命令

命令コード	ニーモニック	動作
10000dddxxxxxxxx	BNEZ *d*, X	レジスタ d が非 0 なら PC に X を足す
10001dddxxxxxxxx	BEQZ *d*, X	レジスタ d が 0 なら PC に X を足す
10010dddxxxxxxxx	BMI *d*, X	レジスタ d が負なら PC に X を足す
10011dddxxxxxxxx	BPL *d*, X	レジスタ d が非負なら PC に X を足す

命令コード	ニーモニック	動作
11000xxxxxxxxxx	Jump	無条件に PC に X を足して相対ジャンプ

RISC16 の動作（1/2）

LD/ST 系の命令は 4 状態，その他の命令は 3 状態で実行する．

1. 命令フェッチ（IF）
   PC で指すアドレスから IR へ命令をフェッチ．同時に PC に 2 を加算．
2. レジスタフェッチ（RF）
   IR のレジスタフィールドで指定された 2 つのレジスタの値を treg1 と treg2 に読み出す．2 つ読み出す必要が無い場合でも害はないので読み出す．
3. 実行（EX）
   ALU で演算を行い IR のフィールドで指定されたレジスタへ書き戻す．LD 命令では treg2 を addr に出力してメモリへのアクセスを行う．ST 命令では treg1 を ALU をスルーして WDR に書き込む．

RISC16 の動作（2/2）

4. 書き戻し（WB）
   LD 命令では RDR から ALU をスルーしてレジスタファイルへ.
   ST 命令では treg2 を addr に出力して WDR の内容をメモリへ書き込む.

分岐命令の動作

1. 命令フェッチ
   同じ
2. レジスタフェッチ
   同じ
3. 実行
   - sbus1 に PC の値を出力
   - sbus2 に IR の下位 8 ビットを 16 ビットに符号拡張した値を出力
     （JUMP 命令では IR の下位 10 ビットを 16 ビットに符号拡張）
   - ALU の op を加算に設定
   - treg1 が分岐条件を満たすならば PC に加算の結果を取り込み

メモリマップド I/O

- 入出力（I/O）の充実
  - たとえば加算の結果を 7 セグメント LED に数字で表示したい
  - しかし，「LED に表示する命令」を新たに加えるのは嫌

$$\Downarrow$$

  - メモリマップド I/O
    - それぞれの入出力デバイスにアドレスを割り当て
    - 普通の LD 命令と ST 命令を使う
- 今回のシステムでは $200_{(16)}$ 番地に ST するとデータが 7 セグメント LED に表示される
  - もはや $200_{(16)}$ 番地に記憶させることはできない
  - $200_{(16)}$ 番地からデータをロードした場合の動作は不定

乗算プログラムの例

```
@00 00001011 00100000 // LLI r3, #0x20
@02 00000000 01110001 // LD r0, (r3)
@04 00001100 00100010 // LLI r4, #0x22
@06 00000001 10010001 // LD r1, (r4)
@08 00001101 00100100 // LLI r5, #0x24
@0a 00000010 10110001 // LD r2, (r5)
@0c 00000010 00000100 // ADD r2, r0
@0e 00100001 11111111 // ADDI r1, #-1
@10 10000001 11111010 // BNEZ r1, #-6
@12 00000010 10110000 // ST r2, (r5)
@14 00110110 00000010 // LUI r6, #0x02
@16 00000010 11010000 // ST r2, (r6)
@18 11000111 11111110 // JMP #-2
@1a 00000000 00000000 //
@1c 00000000 00000000 //
@1e 00000000 00000000 //
@20 00010010 00110100 // m
@22 00000000 00000100 // n
@24 00000000 00000000 // m * n
```

LUI 命令

```
@14 00110110 00000010 // LUI r6, #0x02
@16 00000010 11010000 // ST r2, (r6)
```

- LED に表示させるため r6 に $200_{(16)}$ を設定したい
- 2 進数で 8 ビットを超えるため LLI 命令では設定できない
- LUI 命令で $2_{(16)}$ を設定
  - $\Rightarrow$ 左 8 ビットシフトされて $200_{(16)}$ が格納される

実行例

```
# ==== clock: 67 state: WB ====
#  pc:0018 ir:0000001011010000 treg1:48d0 treg2:0200 rdr:0000 wdr:48d0 oe:0 we:1
#  sbus1:xxxx sbus2:0000 dbus:0000 addr:0200 din:xxxx dout:48d0 led:000000
#  regs: 1234 0000 48d0 0020 0022 0024 0200 0000
#  mem[00-07]: 0b 20 00 71 0c 22 01 91
#  mem[08-0f]: 0d 24 02 b1 02 04 21 ff
#  mem[10-17]: 81 fa 02 b0 36 02 02 d0
#  mem[18-1f]: c7 fe 00 00 00 00 00 00
#  mem[20-27]: 12 34 00 04 48 d0 00 00
#
# ==== clock: 68 state: IF ====
#  pc:0018 ir:0000001011010000 treg1:48d0 treg2:0200 rdr:0000 wdr:48d0 oe:1 we:0
#  sbus1:0018 sbus2:0002 dbus:001a addr:0018 din:c7fe dout:xxxx led:0048d0
#  regs: 1234 0000 48d0 0020 0022 0024 0200 0000
#  mem[00-07]: 0b 20 00 71 0c 22 01 91
#  mem[08-0f]: 0d 24 02 b1 02 04 21 ff
#  mem[10-17]: 81 fa 02 b0 36 02 02 d0
#  mem[18-1f]: c7 fe 00 00 00 00 00 00
#  mem[20-27]: 12 34 00 04 48 d0 00 00
```