マイクロプロセッサ演習

2004 年度

第 9 回

1　はじめに〜データパスと制御

今回から教科書第五章「プロセッサ:データパスと制御」について学ぶ。プロセッサは大きく分けて 2 つの要素で構成される。データパス (datapath) と制御 (control) である。教科書によるとデータパスはプロセッサの筋肉 (brawn) であり、制御は脳 (brain) であるとされている。データパスは datapath という綴りにあるようにデータ経路のことであり、この経路に沿って加算やロード /ストアなど、実際の演算処理が行われる。制御は命令に従って何をすればよいか、データパスやメモリなどに指示を出す。

主に以下の流れで解説を進める。

1. 単一サイクルのデータパス (今回)
2. マルチサイクルのデータパス (次回)
3. パイプラインを用いたデータパス (教科書下巻、授業・演習の範囲外)

三つ目の「パイプラインを用いたデータパス」は授業の範囲外であるが、現代のプロセッサではパイプラインの考え方は必須であるため、興味のある人は自習するとよいだろう。

今回学ぶ「単一サイクルのデータパス」の動作の模式図を図 1 に示す。すなわち、1 クロックにつき 1 命令を実行するような方式である。なお、今回の教科書第五章は非常に広範な内容を含んでいるため、この演習だけで全てを解説するのは時間的に難しいかもしれない。そのため、本資料は教科書や授業の内容を理解するためのガイドとして利用することを意図して作成することにする。教科書の解説は非常に丁寧に書かれているので、時間をかけてじっくり読んでいけば必ず理解できるはずである。

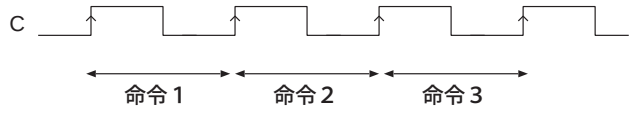

図 1: 単一サイクルで命令を実行する。

まず、MIPS 命令セットの実現方式の概念的な構造図を表したのが図 2 である。これはまだ不完全であるが、これに肉付けをすることで単一サイクルのデータパスが構成されるため、まずこの図の理解を足掛かりにしよう。以下、順に解説しよう。

1. 過去の授業と演習において、プロセッサはメモリに格納された命令を一命令ずつ実行することでプログラムの機能が実現されることを学んで来た。それに対応し、まず PC (プログラムカウンタ) に格納されたメモリアドレスを読み込み、命令メモリから命令を取得する (図中 (a))。

2. この命令が例えば add 命令であれば「add $t0, $t1, $t2」のように 3 つのレジスタを「読み込み用 2 つ、書き込み用 1 つ」として指定することになる。そのため、命令メモリからレジスタへ 3 つの入力が入る (図中 (b))。

   一方、この命令が lw 命令「lw $t0, 4($t1)」であれば、「読み出し用 1 つ、書き込み用 1 つ」の計 2 つの入力がレジスタに入る。さらに、ベースアドレスからのずれの値「4」を後段の ALU に入力する必要がある (図中 (c))。

3. いま、命令が 「add $t0, $t1, $t2」であれば、レジスタからの出力は 2 つ ($t1 の値と $t2 の値) である。この値を ALU に入力して加算を行う (図中 (d))。

   一方、命令が「lw $t0, 4($t1)」であれば、「$t1 の値に 4 を足す」演算が必要であり、これも ALU で行われる (図中 (e))。

4. ALU の計算結果は、命令が 「add $t0, $t1, $t2」であればそのままレジスタにデータとして入力され、レジスタ $t0 に書き込まれる (図

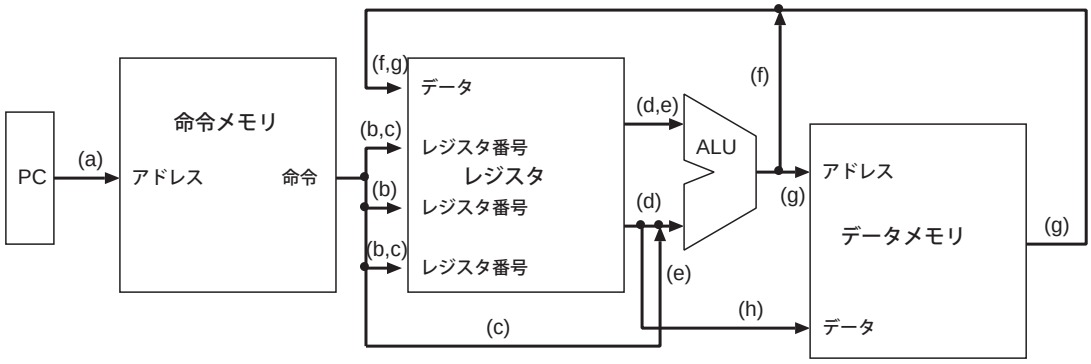

図 2: MIPS の命令の実現方式の概念図。

中 (f))。一方、命令が「lw $t0, 4($t1)」であれば、加算結果を元にデータメモリから値を読み込んでレジスタ $t0 に格納する必要がある (図中 (g))。

5. なお、レジスタからデータメモリの「データ」入力への延びる線は、例えば「sw $t0, 4($t1)」命令で使われる。レジスタ $t0 の値をデータメモリに格納するために使われる (図中 (h))。

以上が図 2 でのデータの概念的な流れである。この概念を一つ一つ実現することでデータパスが完成する。

2 データパス構成の準備 1

まずデータパスを構成する論理要素について触れておこう。MIPS の具体的な機能ユニットには 2 種類の論理要素がある。一つは状態を保持できる**状態論理要素**、もう一つはデータの値に何らかの操作を行う**組み合わせ論理要素**である。論理回路の授業で、状態論理要素の例としてフリップフロップ、組み合わせ論理要素の例として加算器等を学んだことを覚えているだろうか。

図 2 の要素を、この 2 つに分類すると以下の様になる。なお、データパスで多く使われるマルチプレクサについても分類に加えた (マルチプレクサに関しては演習第八回参照)。

- 状態論理要素:PC、命令メモリ、レジスタ、データメモリ
- 組合せ論理要素：ALU、マルチプレクサ

状態論理要素がその保持する値を更新するタイミングには、さまざまな方式がある (例えば、教科書下巻付録 B を参照) が、以下では**エッジ・トリガ・クロック方式** を中心に話を進める。なお、クロックエッジには立ち上がりおよび立ち下がりがあり、それぞれに応答する状態論理要素が考えられる。

イメージをつかむため、クロックエッジをトリガとする D フリップフロップの動作を図 3 に示した。立ち上がりエッジと立ち下がりエッジをトリガとした動作をそれぞれ示した。

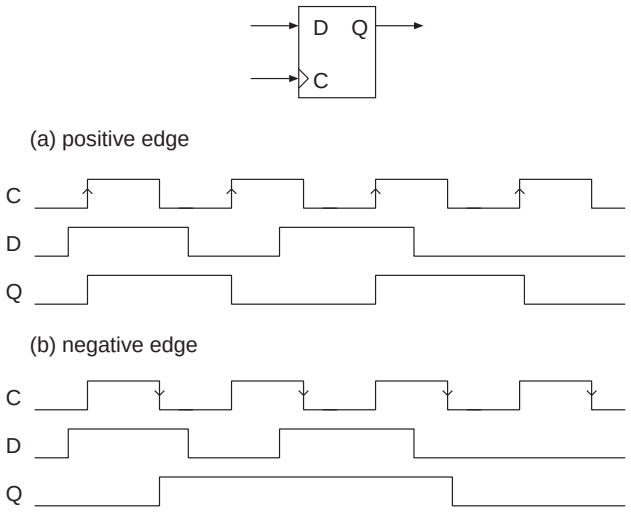

図 3: クロックエッジをトリガとする D フリップフロップの動作。(a) 立ち上がりエッジをトリガとする場合 (b) 立ち下がりエッジをトリガとする場合。どちらの動作となるかは D フリップフロップの構成により異なる。

このようなクロック・エッジ・トリガ方式で状態の更新を行う状態論理要素と、組合せ論理要素を組

み合わせると、図 4 のようなデータ処理方式が可能になる。図 4(a) では、状態論理要素に保持される

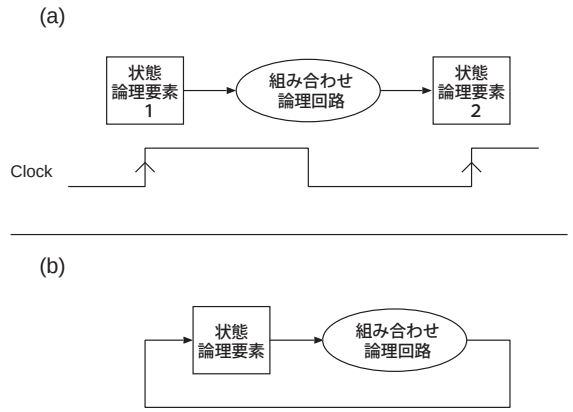

図 4: 状態論理要素と組み合わせ論理回路の組合せによるデータ処理。

値の更新はクロックの立ち上がりで行ない、その間に組合せ論理回路にて演算を行なう、という方式である。同様に、図 4(b) のようにフィードバックがある場合も正しく動作する。図 4(a) は次回の「マルチサイクルのデータパス」で用いられる手法であり、図 4(b) は今回の「単一サイクルのデータパス」で用いられる手法である。なお、いずれの場合も、クロックサイクル時間を短縮する際に考慮すべき主要な要素は、組合せ論理回路の部分の演算にかかる遅延時間であることに注意しよう。

3 データパス構成の準備 2 〜レジスタファイル

本章では、データパス構築の際に頻出するレジスタファイルの構造と動作を解説する。レジスタファイルとはレジスタが複数集まったものである (MIPS では 32 個)。なお、本章は教科書下巻「付録 B 論理設計の基礎」に基づいている。

MIPS で用いられるレジスタファイルを図 5 に示した。2 つの読み出しポートと 1 つの書き込みポート、およびクロックとして用いられる書き込み用の制御信号 (Write) があることがわかる。

読み出し用ポートとその構成を図 6 に示した。32 個あるレジスタの番号を指定すると、そのレジスタの内容が出力に現れる。図にあるように、レジスタの読み出しデータは組合せ論理回路のように、レジ

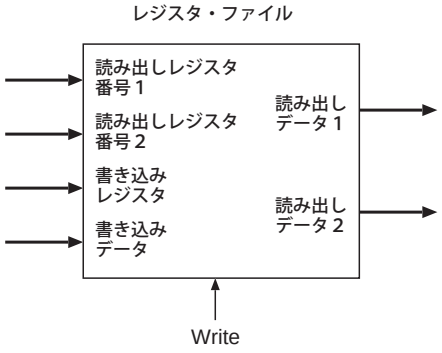

図 5: MIPS で用いられるレジスタファイル。2 つの読み出しポートと 1 つの書き込みポート、およびクロックとして用いられる書き込み用の制御信号 (Write) がある。

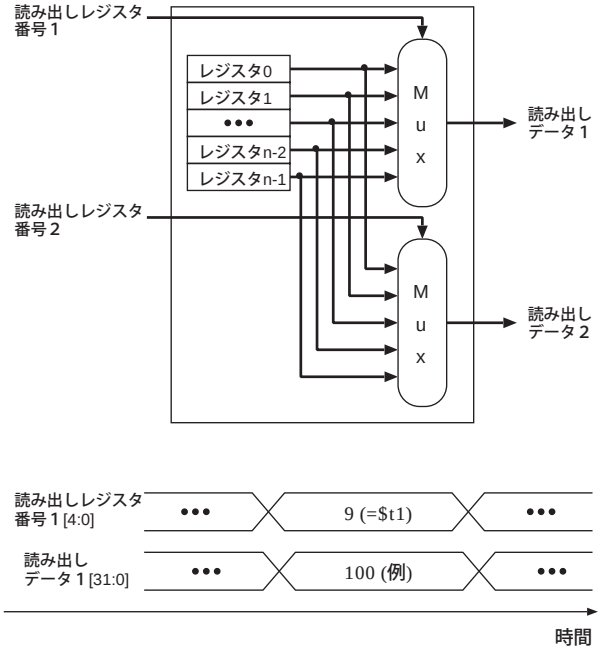

図 6: レジスタの読み出しポート。

スタ番号の変更に伴い、即座に反映される (もちろん、マルチプレクサ (Mux) 分の遅延は伴う)。このように、**レジスタの読み出しは組合せ論理回路の様に読み出せる**ことに注意しよう。

一方、レジスタの書き込みポートは図 7 の様に構成される。レジスタ番号 (0 〜 31) から、デコーダを利用して一つのレジスタのみを選択し、そこにレジスタデータ D (32 ビット) を書き込むようになっている。また、一つ一つのレジスタは 32 個の D フリップフロップから構成される。**レジスタファイル**

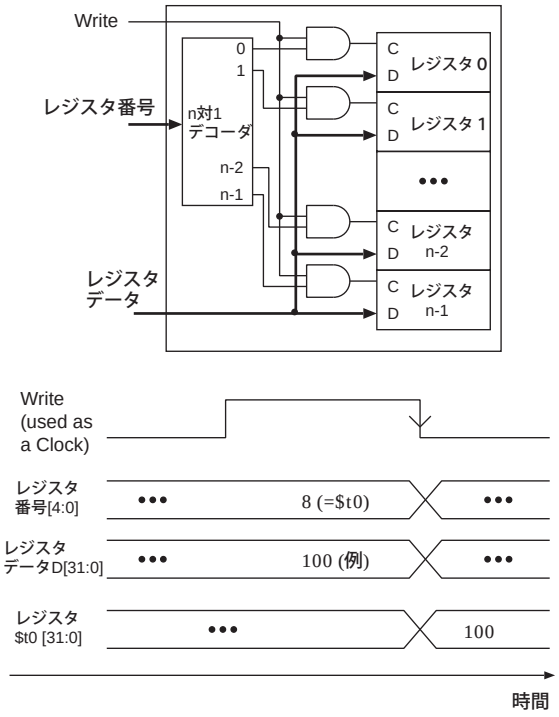

図 7: レジスタ・ファイルの書き込みポート。D フリップフロップは、クロックの立ち下がりエッジでトリガされるとして図を描いた。

の書き込みは状態論理要素として利用されることに注意しよう。

4 データパスの構築と制御

以上の準備のもと、「単一サイクルで動作するデータパス」の構築に進む (教科書 317 ページ)。様々な命令 (and, lw, sw, jr 等) を考え、それらが正しく動作するよう、論理要素を追加してゆく。ここで全ての機能追加手順を解説するのは難しいので、導入を行なうに留め、残りは教科書や授業資料を元に学ぶという方法を取る。

[命令フェッチ部]

まず、**命令フェッチ部**を構築する。プログラムの現在の命令の位置 (メモリアドレス) は PC (Program Counter) に格納されていることを第七回「I 形式」の章で学んだ。この PC の値に基づいて命令メモリから命令が取得される (フェッチされる) のは図 2 で見た通りである。アドレス PC から命令が取得された後、PC の値は次の命令を指すため +4 されなければならない。それを付加したのが図 8 である。PC は状態論理要素であるので、PC+4 の値は次の

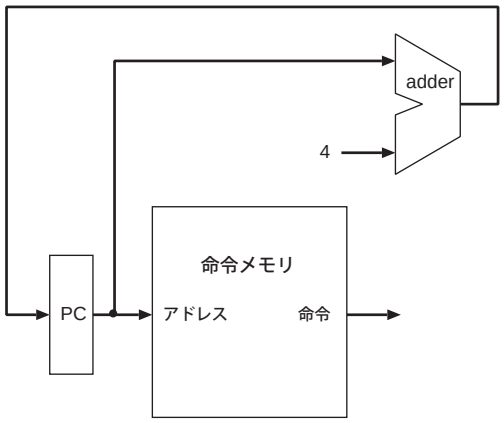

図 8: データパスのうち、命令をフェッチしてプログラム・カウンタを繰り上げる部分。

クロックの立ち上がりで PC に書き込まれる。図 8 の PC と加算器 (adder) からなるフィードバックは図 4(b) に対応することに注意しよう。すなわち、PC が状態論理要素であり、adder は組合せ論理回路である。なお、レジスタ同様、命令メモリも読み出しは組合せ論理回路のように読み出すことができる。

[R 形式命令とロード /ストア命令部]

さらに、データパスに R 形式命令 (add, sub 等) とロード /ストア命令を実行できるよう手を加えよう。まず、R 形式命令を実行するため、命令の「15〜11 ビット」を書き込みレジスタ番号として、また「20〜16 ビット」および「25〜21 ビット」を読み出しレジスタ番号として分離する (第七回「R 形式」の章参照)。また、「lw $t0, 4($t1)」の 4 を ALU に入力する際、この 4 は 16 ビットなので (第七回「I 形式」の章参照) 32 ビットに符号拡張する必要がある (符号拡張は第八回補足参照)。

また、ALU への入力を選択するマルチプレクサが必要である。例えば「add $t0, $t1, $t2」という命令であれば、「レジスタ $t1, $t2 の内容」が入力となり、「lw $t0, 4($t1)」という命令であれば「レジスタ $t1 の内容と符号拡張された 4」が入力とならねばならない。さらに、このマルチプレクサへの制御入力 ALUSrc (赤字) が必要である。例えば、ALUSrc が 0 の時にレジスタからの入力が、 1 のときに符号拡張からの入力が選ばれるようにする。

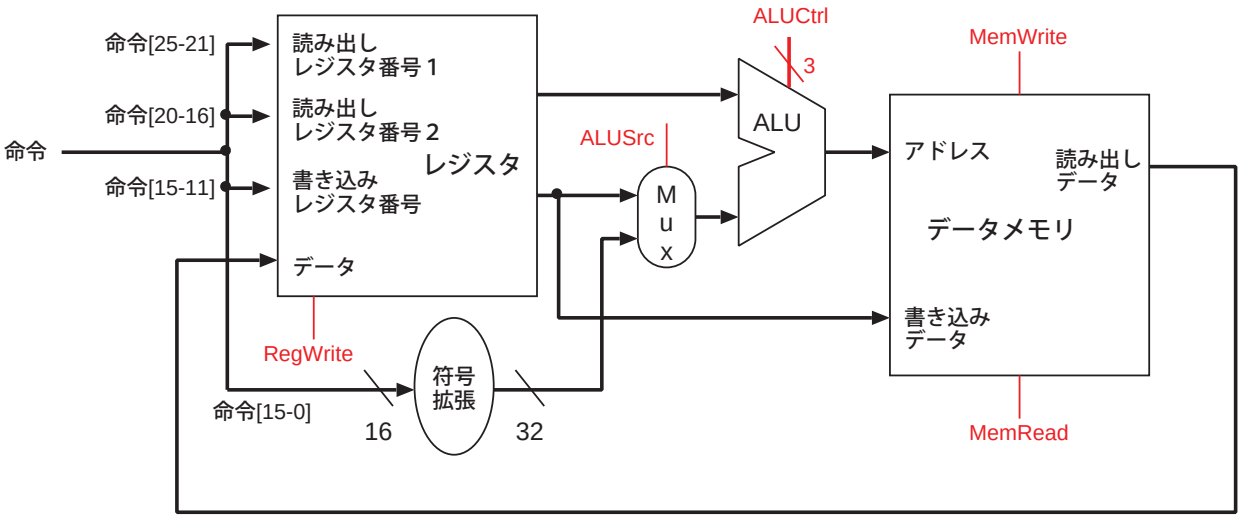

図 9: データパスのうち、R 形式命令とロード /ストア命令を実行する部分。

[**残りのデータパス**]
以上、命令フェッチ部と R 命令、ロード /ストア命令実行部のデータパスの構成をみた。残りは教科書や授業資料を参考に、単一サイクルのデータパスの構成を理解して欲しい。

[**制御**]
さて、図 9 において、赤い文字で書かれた信号がいくつか存在する。これらが、データパスをコントロールするための**制御入力** である。

例えば先程触れた ALUSrc であれば、R 形式の命令であれば 0 を、ロード /ストア命令であれば 1 を出力するよう制御信号を出さねばならない。この ALUSrc の生成には、フェッチした命令の Op コードを元にする (Op コードについては第七回参照)。R 形式、およびロード /ストア命令の Op コードとそれに対する制御信号 ALUSrc を表 1 に示した。このような真理値表を元に、必要な制御信号を実現する論理回路を設計することができる。(この表は R 形式とロード /ストア命令のみしか考慮しなかったが、実際にはより多くの命令を判定しなければならない)

5 問題

[**問 1**]
命令「lw $t0, 4($s1)」を実行するときに働くデータパスに○印をつけ、どのフェーズで動作するか脇に下の番号をつけなさい。図が不鮮明な場合は教科書 p.332 の図 5.19 を参照。

1. 命令フェッチ
2. レジスタ読み取り
3. ALU 演算
4. レジスタ書き込み
5. メモリ read/write

[**問 2**]
jr 命令を実行するために必要なデータパスを図に追加しなさい。

[**問 3**]
テキスト p.327 図 5.15 の ALU 制御コードの真理値表から ALU の制御信号の ALUCont2 を生成する回路を合成しなさい。なお、ALUCont2 は教科書「ALU 制御入力」の最上位ビットである。

(ヒント) 入力信号数が 8 なのでカルノー図を書くのは困難。ALUCont2 が 1 になる条件を直接論理式で求める。don't care が多いので用意に求められる。簡略化は完全でなくても良い。

命令名	Op5	Op4	Op3	Op2	Op1	Op0	ALUSrc
R 形式	0	0	0	0	0	0	0
lw	1	0	0	0	1	1	1
sw	1	0	1	0	1	1	1

表 1: 制御信号 ALUSrc を判定するための真理値表。

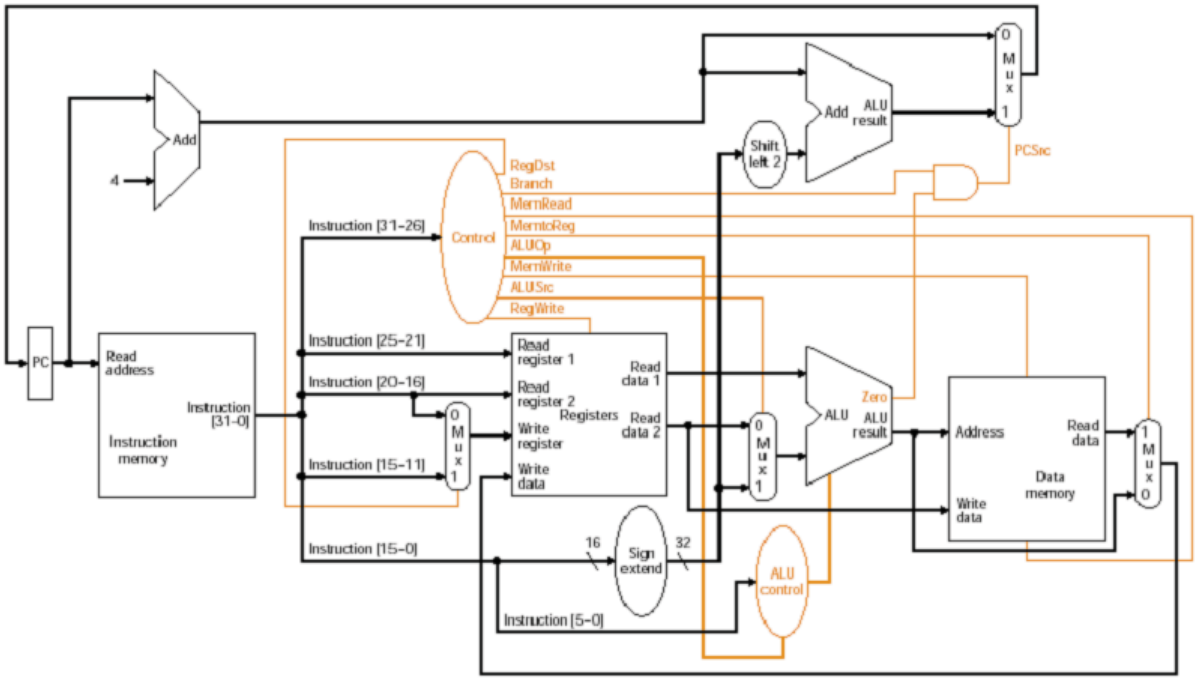

図 10: 問 1 のための図

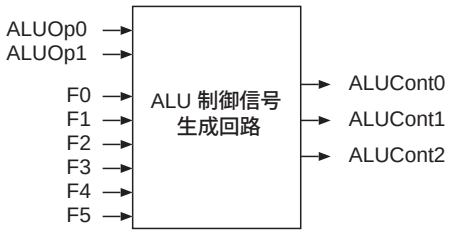

図 12: ALU 制御信号生成回路

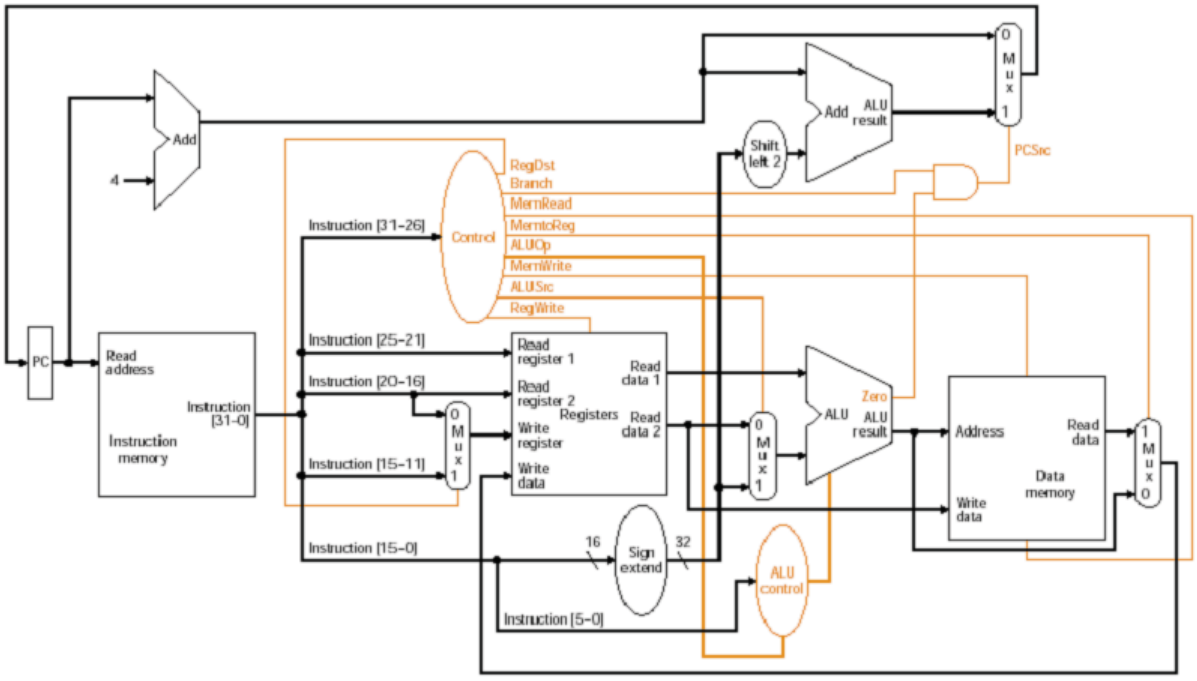
Add
4
M
u
x
0
1
Add
ALU
result
Shift
left 2
PCSrc
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31-26]
PC
Read
address
Instruction
[31-0]
Instruction
memory
Instruction [25-21]
Instruction [20-16]
Instruction [15-11]
Instruction [15-0]
Instruction [5-0]
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
16
Sign
extend
32
ALU
control
Zero
ALU
ALU
result
Address
Read
data
Data
memory
Write
data

図 11: 問 2 のための図