計算機設計論 レポート課題:MIPS プロセッサの回路設計

1295149 森岡悠人

2025 年 8 月 16 日

1　モジュール仕様書

Quartus の RTL Viewer を用いて出力した，DE10-lite に書き込んだモジュールのブロック図を図 1 に示す.

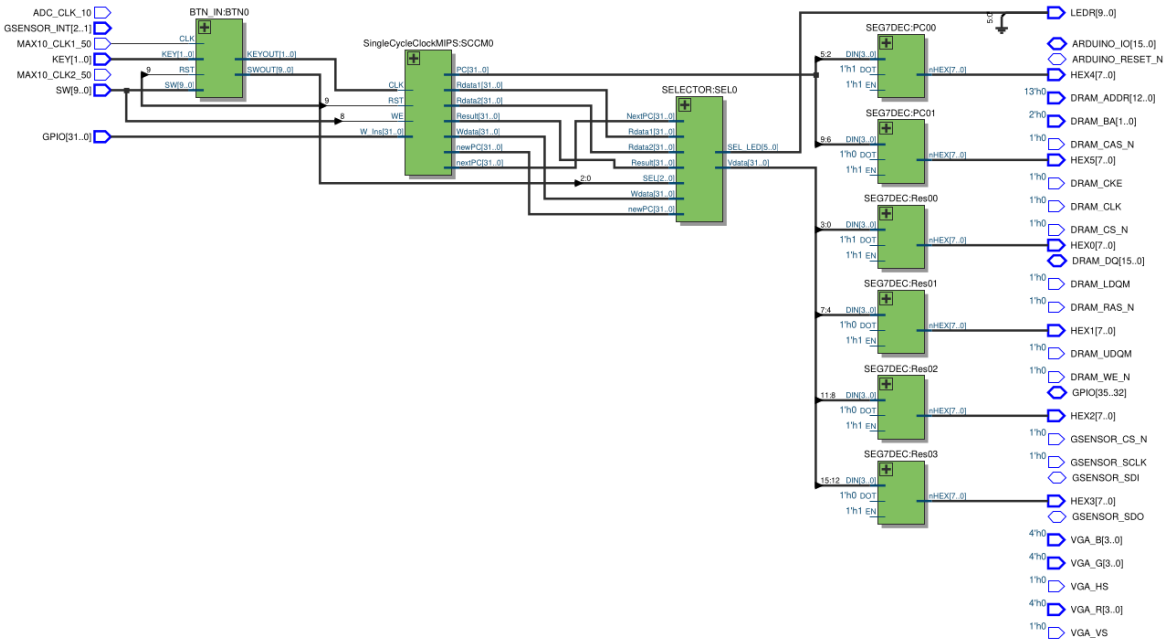

図 1　作成した MIPS 回路のブロック図

2　動作検証

作成した verilog コードが MIPS の命令セットを実行できるかどうかの検証を行った．テストプログラムとして，教科書 [1] に掲載されているアセンブラプログラム（load_store, arithmetic, array, if_then_else, while, function, recursion, hanoi）を用いた．プログラムは CPUlator MIPS System Simulator [2] を用いてコンパイルし，32bit の 16 進数で出力されたバイナリを `IMem.txt` に書き込んでおき，IM に読み込ませて実行し

た．動作の流れとデータメモリの中身を確認するため，modelsim20.1 を用いて動作のシミュレーションと検証を行った．シミュレーション結果は，display 命令を用いて，PC, Instruction, ALU_result レジスタの順番が期待通りかどうかを確認した．シミュレーションの様子を図 2 に示す．

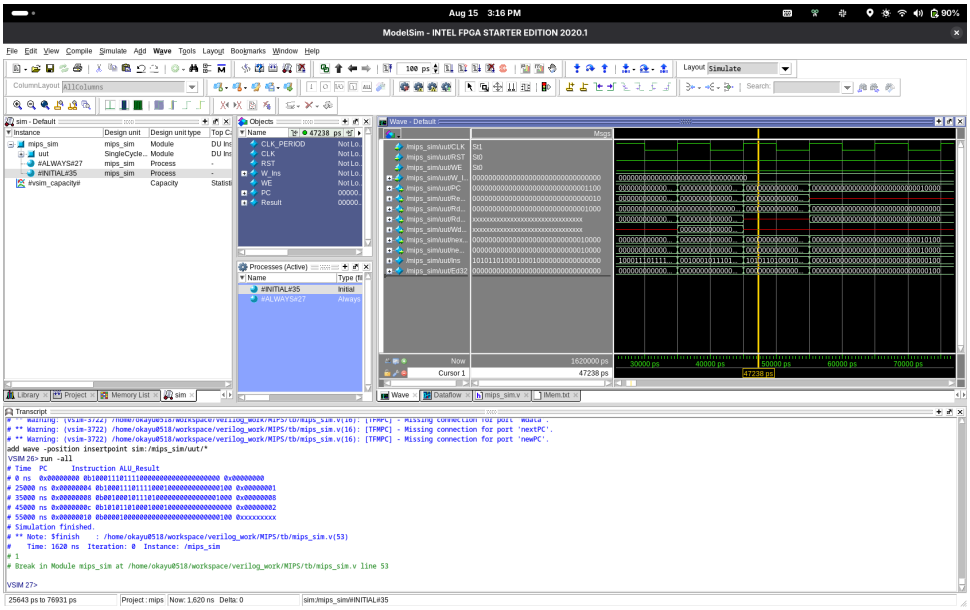

図 2　modelsim によるシミュレーションの様子

2.1　test: load_store

2.1.1　load_store のテスト結果

```
run -all
# Time PC Instruction ALU_Result
# 0 ns 0x00000000 0b10001110111100000000000000000000 0x00000000
# 25000 ns 0x00000004 0b10001110111100010000000000000100 0x00000001
# 35000 ns 0x00000008 0b00100010111010000000000000001000 0x00000008
# 45000 ns 0x0000000c 0b10101101000100010000000000000000 0x00000002
# 55000 ns 0x00000010 0b00001000000000000000000000000100 0xxxxxxxxx
# Simulation finished.
```

2.2　test: arithmetic

あらかじめテストベンチで DMem の 0 番地からそれぞれ変数 a=1,b=2,c=3,d=0 と変数のオフセットを保持するレジスタ$s7=0 を初期値として設定してシミュレーションを実行した．その結果，期待通り a, b, c の値が add 命令で足し合わされ，$s3=6 であることを確認した．

2.2.1 arithmetic のテスト結果

```
run -all
# Time PC Instruction ALU_Result
# 0 ns 0x00000000 0b10001110111100000000000000000000 0x00000000
# 25000 ns 0x00000004 0b10001110111100010000000000000100 0x00000001
# 35000 ns 0x00000008 0b10001110111100100000000000001000 0x00000002
# 45000 ns 0x0000000c 0b00000010000100010100000000100000 0x00000003
# 55000 ns 0x00000010 0b00000001000100101001100000100000 0x00000006
# 65000 ns 0x00000014 0b00001000000000000000000000000101 0xxxxxxxxx
# ==== Simulation Results ====
# $s3 register (R19) final value: 0x00000006
# Simulation finished.
```

2.3 test: array

あらかじめテストベンチで DMem の 0 番地からそれぞれ配列 a[0]=0, a[1]=1..., a[9]=9 と配列のオフセットを保持するレジスタ$s7=0 と$s2=2 を初期値として設定してシミュレーションを実行した．その結果，期待通り$s1=2, $s2=5 であることを確認した．

2.3.1 array のテスト結果

```
run -all
# Time PC Instruction ALU_Result
# 0 ns 0x00000000 0b00000000000100000100000010000000 0x00000008
# 25000 ns 0x00000004 0b00000010111010000100000000100000 0x00000008
# 35000 ns 0x00000008 0b10001101000100010000000000000000 0x00000002
# 45000 ns 0x0000000c 0b10001110111100100000000000010100 0x00000005
# 55000 ns 0x00000010 0b00001000000000000000000000000100 0xxxxxxxxx
# ==== Simulation Results ====
# $s1 register (R17) final value: 0x00000002
# $s2 register (R18) final value: 0x00000005
# Simulation finished.
```

2.4 test: if_then_else

まず，if 文が真になる場合のテストを行った．あらかじめテストベンチでレジスタ$s0=0xa, $s1=0xa, $s2=0x0 を初期値として設定してシミュレーションを実行した．その結果，$s0 と$s1 が等しいため，$s2 に$s1 の値が代入され，$s2=0xa であることを確認した．

2.4.1 if_then_else のテスト結果 (真の場合)

```
run -all
# Time PC Instruction ALU_Result
# 0 ns 0x00000000 0b00010010000100010000000000000010 0x00000000
# 25000 ns 0x0000000c 0b00000000000100011001000000100000 0x0000000a
# 35000 ns 0x00000010 0b00001000000000000000000000000100 0xxxxxxxxx
# ==== Simulation Results ====
# $s0 register (R16) final value: 0x0000000a
# $s1 register (R17) final value: 0x0000000a
# $s2 register (R18) final value: 0x0000000a
# Simulation finished.
```

次に，if 文が偽になる場合のテストを行った．あらかじめテストベンチでレジスタ$s0=0xa，$s1=0xb，$s2=0x0 を初期値として設定してシミュレーションを実行した．その結果，$s0 と$s1 が等しくないため，$s2 に$s0 の値が代入され，$s2=0xa であることを確認した．

2.4.2 if_then_else のテスト結果 (偽の場合)

```
run -all
# Time PC Instruction ALU_Result
# 0 ns 0x00000000 0b00010010000100010000000000000010 0xffffffff
# 25000 ns 0x00000004 0b00000000000100001001000000100000 0x0000000a
# 35000 ns 0x00000008 0b00001000000000000000000000000100 0xxxxxxxxx
# 45000 ns 0x00000010 0b00001000000000000000000000000100 0xxxxxxxxx
# ==== Simulation Results ====
# $s0 register (R16) final value: 0x0000000a
# $s1 register (R17) final value: 0x0000000b
# $s2 register (R18) final value: 0x0000000a
# Simulation finished.
```

真の場合と偽の場合の PC の遷移を比べると，偽の場合は else 節のラベルを飛び越えるために J 命令が実行されているためことがわかる．そのため偽の場合は 1 命令多い．

2.5 test: while

あらかじめテストベンチで配列 a[10] を 0 で初期化し，配列のオフセットを保持するレジスタ$s7=0, $s0=0 を初期値として設定してシミュレーションを実行した．その結果，$s0 が 10 になるまで while 文が繰り返され，配列 a[0] から a[9] にそれぞれ 0 から 9 までの値が代入されていることを確認した．

2.5.1 while のテスト結果

```
run -all
```

```
# Time PC Instruction ALU_Result
# 0 ns 0x00000000 0b00101010000010000000000000001010 0x00000001
# 25000 ns 0x00000004 0b00010001000000000000000000000101 0x00000001
# 35000 ns 0x00000008 0b00000000000100000100000010000000 0x00000000
# 45000 ns 0x0000000c 0b00000010111010000100000000100000 0x00000000
# 55000 ns 0x00000010 0b10101101000100000000000000000000 0x00000000
# 65000 ns 0x00000014 0b00100010000100000000000000000001 0x00000001
# 75000 ns 0x00000018 0b00001000000000000000000000000000 0xxxxxxxxx
# 85000 ns 0x00000000 0b00101010000010000000000000001010 0x00000001
# 95000 ns 0x00000004 0b00010001000000000000000000000101 0x00000001
# 105000 ns 0x00000008 0b00000000000100000100000010000000 0x00000004
# 115000 ns 0x0000000c 0b00000010111010000100000000100000 0x00000004
# 125000 ns 0x00000010 0b10101101000100000000000000000000 0x00000001
# 135000 ns 0x00000014 0b00100010000100000000000000000001 0x00000002
# 145000 ns 0x00000018 0b00001000000000000000000000000000 0xxxxxxxxx
# 155000 ns 0x00000000 0b00101010000010000000000000001010 0x00000001
# 165000 ns 0x00000004 0b00010001000000000000000000000101 0x00000001
# 175000 ns 0x00000008 0b00000000000100000100000010000000 0x00000008
# 185000 ns 0x0000000c 0b00000010111010000100000000100000 0x00000008
# 195000 ns 0x00000010 0b10101101000100000000000000000000 0x00000002
# 205000 ns 0x00000014 0b00100010000100000000000000000001 0x00000003
# 215000 ns 0x00000018 0b00001000000000000000000000000000 0xxxxxxxxx
# 225000 ns 0x00000000 0b00101010000010000000000000001010 0x00000001
# 235000 ns 0x00000004 0b00010001000000000000000000000101 0x00000001
# 245000 ns 0x00000008 0b00000000000100000100000010000000 0x0000000c
# 255000 ns 0x0000000c 0b00000010111010000100000000100000 0x0000000c
# 265000 ns 0x00000010 0b10101101000100000000000000000000 0x00000003
# 275000 ns 0x00000014 0b00100010000100000000000000000001 0x00000004
# 285000 ns 0x00000018 0b00001000000000000000000000000000 0xxxxxxxxx
# 295000 ns 0x00000000 0b00101010000010000000000000001010 0x00000001
# 305000 ns 0x00000004 0b00010001000000000000000000000101 0x00000001
# 315000 ns 0x00000008 0b00000000000100000100000010000000 0x00000010
# 325000 ns 0x0000000c 0b00000010111010000100000000100000 0x00000010
# 335000 ns 0x00000010 0b10101101000100000000000000000000 0x00000004
# 345000 ns 0x00000014 0b00100010000100000000000000000001 0x00000005
# 355000 ns 0x00000018 0b00001000000000000000000000000000 0xxxxxxxxx
# 365000 ns 0x00000000 0b00101010000010000000000000001010 0x00000001
# 375000 ns 0x00000004 0b00010001000000000000000000000101 0x00000001
# 385000 ns 0x00000008 0b00000000000100000100000010000000 0x00000014
# 395000 ns 0x0000000c 0b00000010111010000100000000100000 0x00000014
```

```
# 405000 ns 0x00000010 0b10101101000100000000000000000000 0x00000005
# 415000 ns 0x00000014 0b00100010000100000000000000000001 0x00000006
# 425000 ns 0x00000018 0b00001000000000000000000000000000 0xxxxxxxxx
# 435000 ns 0x00000000 0b00101010000010000000000000001010 0x00000001
# 445000 ns 0x00000004 0b00010001000000000000000000000101 0x00000001
# 455000 ns 0x00000008 0b00000000000100000100000010000000 0x00000018
# 465000 ns 0x0000000c 0b00000010111010000100000000100000 0x00000018
# 475000 ns 0x00000010 0b10101101000100000000000000000000 0x00000006
# 485000 ns 0x00000014 0b00100010000100000000000000000001 0x00000007
# 495000 ns 0x00000018 0b00001000000000000000000000000000 0xxxxxxxxx
# 505000 ns 0x00000000 0b00101010000010000000000000001010 0x00000001
# 515000 ns 0x00000004 0b00010001000000000000000000000101 0x00000001
# 525000 ns 0x00000008 0b00000000000100000100000010000000 0x0000001c
# 535000 ns 0x0000000c 0b00000010111010000100000000100000 0x0000001c
# 545000 ns 0x00000010 0b10101101000100000000000000000000 0x00000007
# 555000 ns 0x00000014 0b00100010000100000000000000000001 0x00000008
# 565000 ns 0x00000018 0b00001000000000000000000000000000 0xxxxxxxxx
# 575000 ns 0x00000000 0b00101010000010000000000000001010 0x00000001
# 585000 ns 0x00000004 0b00010001000000000000000000000101 0x00000001
# 595000 ns 0x00000008 0b00000000000100000100000010000000 0x00000020
# 605000 ns 0x0000000c 0b00000010111010000100000000100000 0x00000020
# 615000 ns 0x00000010 0b10101101000100000000000000000000 0x00000008
# 625000 ns 0x00000014 0b00100010000100000000000000000001 0x00000009
# 635000 ns 0x00000018 0b00001000000000000000000000000000 0xxxxxxxxx
# 645000 ns 0x00000000 0b00101010000010000000000000001010 0x00000001
# 655000 ns 0x00000004 0b00010001000000000000000000000101 0x00000001
# 665000 ns 0x00000008 0b00000000000100000100000010000000 0x00000024
# 675000 ns 0x0000000c 0b00000010111010000100000000100000 0x00000024
# 685000 ns 0x00000010 0b10101101000100000000000000000000 0x00000009
# 695000 ns 0x00000014 0b00100010000100000000000000000001 0x0000000a
# 705000 ns 0x00000018 0b00001000000000000000000000000000 0xxxxxxxxx
# 715000 ns 0x00000000 0b00101010000010000000000000001010 0x00000000
# 725000 ns 0x00000004 0b00010001000000000000000000000101 0x00000000
# 735000 ns 0x0000001c 0b00001000000000000000000000000111 0xxxxxxxxx
# ==== Simulation Results ====
# REGFILE[0]: 0x00000000
# REGFILE[1]: 0x00000001
# REGFILE[2]: 0x00000002
# REGFILE[3]: 0x00000003
# REGFILE[4]: 0x00000004
```

```
# REGFILE[5]: 0x00000005
# REGFILE[6]: 0x00000006
# REGFILE[7]: 0x00000007
# REGFILE[8]: 0x00000008
# REGFILE[9]: 0x00000009
# Simulation finished.
```

2.6 test: function

2.7 test: recursion

2.8 test: hanoi

hanoi のテストは上記のテストほとんどすべての処理を含むため，Quartus Prime を用いて DE10-Lite に書き込んで実行した．その結果，同様に期待通りの動作を確認できた．

3 考察

4 感想

参考文献

[1] 成瀬正. コンピュータアーキテクチャ. 森北出版, 第 1 版, 2016.

[2] Cpulator mips system simulator. https://cpulator.01xz.net/?sys=mipsr5b. Accessed: 2025-08-16.