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 20 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.

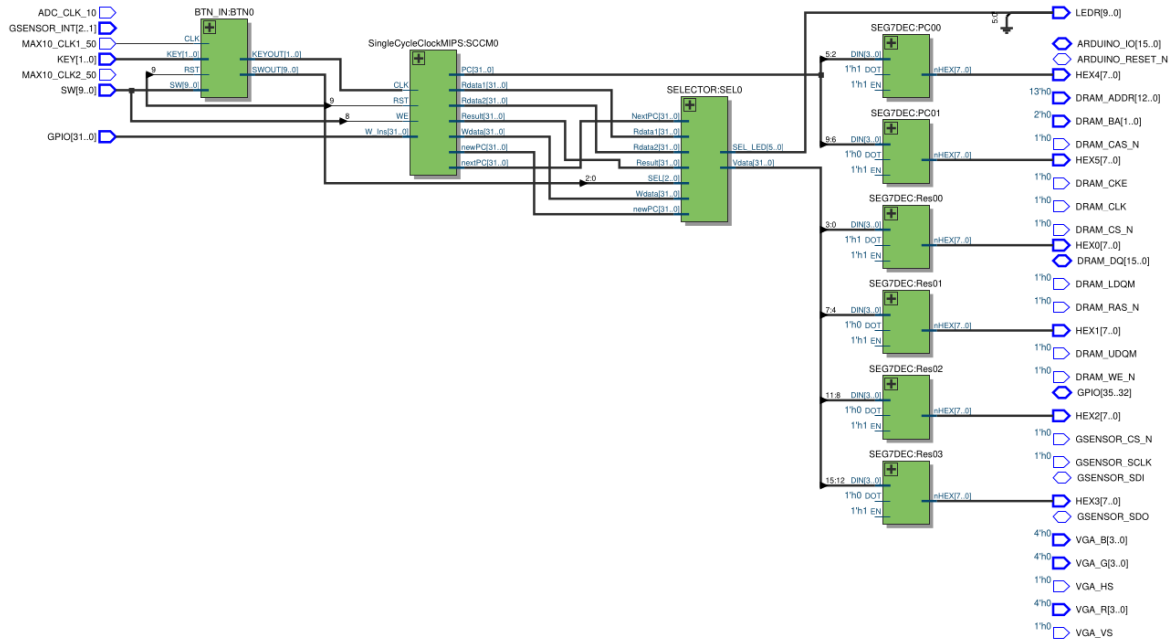

図 1　作成した MIPS 回路のブロック図

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た．その後 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

せた状態でシミュレーションを実行した．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．

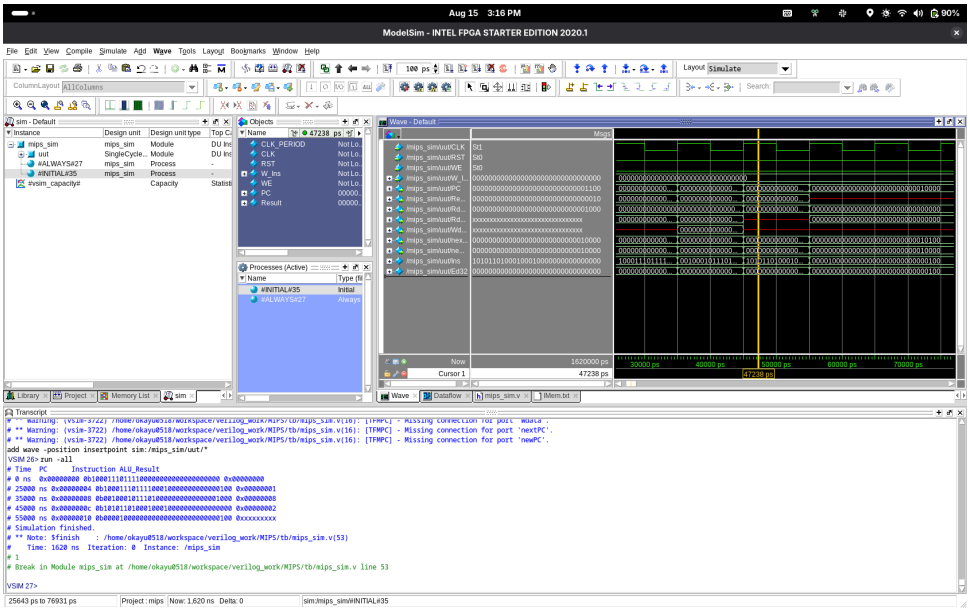

図 2　modelsim によるシミュレーションの様子

2.1　test: load_store

基本的なロード・ストア命令の動作確認を行った．このプログラムではデータメモリから値をロードし、別のアドレスにストアする処理を行う．プログラムの動作は以下の通りである：

- `lw $s0, 0($s7)`: データメモリのアドレス$s7+0 から値をロードし、$s0 に格納
- `lw $s1, 4($s7)`: データメモリのアドレス$s7+4 から値をロードし、$s1 に格納
- `addi $t0, $s7, 8`: $s7+8 をアドレスとして$t0 に計算
- `sw $s1, 0($t0)`: $s1 の値を$t0 が示すアドレスにストア

あらかじめテストベンチで DMem の 0 番地からそれぞれ静的変数 a=10，b=20 と，配列 a[0]=0，配列のオフセットを保持するレジスタ$s7=0 を初期値として設定してシミュレーションを実行した．その結果，期待通り$s0=10, $s1=20, DMem[0]=10 であることを確認した．使用したアセンブリコードとそのテスト結果を付録 A.1 に示す．

2.2　test: arithmetic

基本的な算術演算命令の動作を確認するためのテストを行った．このプログラムは 3 つの値を読み込んで加算を行い、結果をレジスタに格納する．プログラムの動作は以下の通りである：

- 変数 a, b, c の値（1, 2, 3）をそれぞれ`$s0`, `$s1`, `$s2` にロード
- `add $t0, $s0, $s1`: a + b の結果を`$t0` に格納
- `add $s3, $t0, $s2`: (a + b) + c の結果を`$s3` に格納（期待値：6）

あらかじめテストベンチで DMem の 0 番地からそれぞれ静的変数 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．使用したアセンブリコードとそのテスト結果を付録 A.2 に示す．

2.3 test: array

配列のアクセスとインデックス計算の動作を確認するためのテストを行った．プログラムの動作は以下の通りである：

- `sll $t0, $s0, 2`: インデックス`$s0` を 4 倍（左シフト 2 ビット）してワードアドレスに変換
- `add $t0, $s7, $t0`: 配列の基底アドレス`$s7` にオフセットを加算
- `lw $s1, 0($t0)`: 計算されたアドレスから配列要素をロード
- `lw $s2, 20($s7)`: 配列の 5 番目の要素（20 バイトオフセット）をロード

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．使用したアセンブリコードとそのテスト結果を付録 A.3 に示す．

2.4 test: if_then_else

条件分岐命令の動作を確認するためのテストを行った．プログラムの動作は以下の通りである：

- `beq $s0, $s1, L1`: `$s0` と`$s1` が等しい場合は L1 にジャンプ
- 等しくない場合：`$s2 = $s0` を実行して N1 にジャンプ
- 等しい場合（L1）：`$s2 = $s1` を実行

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．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．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．使用したアセンブリコードとそのテスト結果を付録 A.4 に示す．

2.5 test: while

ループ処理と条件判定の動作を確認するためのテストを行った．プログラムの動作は以下の通りである：

- `slti $t0, $s0, 10`: `$s0 < 10` の条件判定結果を`$t0` に格納
- 条件が偽（`$s0 >= 10`）の場合はループを終了して N1 にジャンプ
- 条件が真の場合：配列に`$s0` の値を格納し、`$s0` をインクリメントしてループを継続

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．使用したアセンブリコードとそのテスト結果を付録 A.5 に示す．

2.6 test: function

関数呼び出しとスタック操作の動作を確認するためのテストを行った．このプログラムは 1 から n までの和を計算する関数を実装している．プログラムの動作は以下の通りである：

- メイン部分：引数を`$a0` に設定して sum 関数を呼び出し、戻り値を`$s1` に格納
- sum 関数：スタックにレジスタを退避し、1 から n までの和を計算して`$v0` に結果を返す
- 関数終了時にはスタックからレジスタを復元し、呼び出し元に戻る

あらかじめテストベンチでレジスタ$s0=10 を初期値として設定して 1 から n までの和を求めるプログラムのシミュレーションを実行した．その結果，$s1=0x37=0d55 となることを確認した．使用したアセンブリコードとそのテスト結果を付録 A.6 に示す．

2.7 test: recursion

再帰関数呼び出しの動作を確認するためのテストを行った．このプログラムは再帰を用いて 1 から n までの和を計算する．プログラムの動作は以下の通りである：

- 引数が 1 未満の場合は 0 を返してベースケースとする
- 引数が 1 以上の場合は、引数を 1 減らして再帰呼び出しを行い、その結果に現在の引数値を加算
- 各再帰レベルでスタックに引数と戻りアドレスを保存・復元

2.6 のテストと同様に，あらかじめテストベンチでレジスタ$s0=10 を初期値として設定してシミュレーションを実行した．その結果，$s1=0x37=0d55 となることを確認した．使用したアセンブリコードとそのテスト結果を付録 A.7 に示す．

2.8 test: hanoi

ハノイの塔を解く再帰アルゴリズムを実装し、複雑な再帰処理の動作を確認するためのテストを行った．プログラムの動作は以下の通りである：

- 3 枚の円盤のハノイの塔問題を解く
- `$a0`: 円盤の枚数、`$a1`: 移動元、`$a2`: 移動先、`$a3`: 補助杆
- `$t1`: 移動回数カウンタ（期待値：7 回）
- 再帰の深さに応じてスタックに多くの引数と戻りアドレスを保存

- ベースケース（円盤が 1 枚）では直接移動を実行

円盤が 3 枚のハノイの塔を解くプログラムを実行する．レジスタ$t1 を移動回数カウント用としてシミュレーションを行った．結果，$t1=7 となり，期待通りの動作を確認した．hanoi のテストは算術演算，関数呼び出しなど，上記の多くの処理を含むため，Quartus Prime を用いて DE10-Lite に書き込んで実行した．その結果，modelsim 同様に期待通りの動作をし，最後の無限ループの処理まで到達することを確認できた．使用したアセンブリコードとそのテスト結果を付録 A.8 に示す．

3 考察

3.1 森岡悠人

今回の MIPS 回路設計の課題は松本と協力して課題を分担して取り組んだ．定期的に対面で集まって作業し，問題点やバグを共有しながら進めた．どうしても理解できない部分や，詰まった部分は岩田研究室の M1 に聞きに行くことで解決した．github のリポジトリ上で共同編集を行ったが，回路のソースとテストベンチを分けて開発したため，変更が競合しづらく，効率よく開発を進めることができた．私は作成した回路のテストを担当した．まず，EX.v のテストベンチを書いて各命令を入力したときの ALU の result をチェックし，その後，MIPS 全体のテストベンチを書いて教科書にあるテストプログラムを使って命令実行時の挙動を確認した．テストベンチを作成する際，modelsim の wave だけでは確認するときの効率が悪かったため，display 命令を用いてテスト内容と結果はすべてコンソールに出力させた．また，テスト内容について，生成 AI を用いて効率よくテストを作成することを検討したが，生成 AI は verilog の知識はあまりない様子で，生成したテスト内容はところどころミスがあり，結局すべてのテストを手動で確認して修正した．しかし，テストベンチ全体の構造は生成 AI を用いて作成したため，ある程度効率的にテストベンチを作成できた．このとき，Instruction の値は，微妙な間隔でビットの区切りがあるため，バイナリで書いた．さらに，命令の形式に対応したビットの区切りとなる箇所にはアンダーバーを入れて記述することで読みやすくなるように工夫した．EX.v のテストベンチ全体として，`test_instruction` という task を作成し，そこに各命令のテストとなる入力と期待する出力を入れ，比較することでテストを行った．回路をデバッグする中で意識したことは，回路の出力が期待通りにならないとき，信号の流れを一つずつ追うことである．modelsim のシミュレーションのみたい箇所の信号が確認できる機能を使って，どの信号線まで正常に信号が流れているかを入力から順番に確認することで，問題を特定できた．また，modelsim のデータメモリの中身を確認する機能も sw 命令のデバッグで非常に役に立った．ただし，Online MIPS Simulator では.data 領域に確保するメモリサイズや初期値をを記述すればコンパイルするときに勝手にメモリ上に配置してくれたのに対し，作成した MIPS ではその機能がないため，テストベンチで手動でデータメモリの初期値を設定する必要があった．これは大変だと感じ，既存のコンパイラやアセンブラの自動でメモリに初期値を書き込んでくれる機能のありがたみを感じた．ただし，データメモリに初期値を書き込んだりするのは回路設計の果たして回路レベルでやるべき処理なのか疑問に思った．

今回作成した MIPS の改善点について述べる．まず，FPGA ボードの 7 セグに常に PC をワード単位で表示させたが，これはバイト単位の表示のほうが他シミュレータの結果との比較がしやすいのではないかと考える．また，今回は IMem.txt にバイナリを書き込んでおき，それを読み込む形で命令を実行したが，テストベンチから IM に読み込ませるテキストファイルを指定できるようにできれば，あらかじめテストプログラムをファイル単位で用意しておいて実行できて，テストが捗るのではないかと考えた．しかしそれはそれで，

modelsim にも quartus にも IMem.txt を読み込む処理を別々に書かないといけないため，面倒だなと思った．また，教科書のテストプログラムについて，MULT や DIV など，実装したすべての命令が網羅できているとは言えず，common$_{p}aram.vh$ のすべての命令をテストしようと思うと別で大量のテストベンチを書く必要がある．そのため，もう少し網羅的かつ統一的にテストできるテストプログラムの雛形と正解となる出力データを先生側で用意してもらえるとテストも効率的に行えて，課題の確認もスムーズに行えて良いのではないかと思った．また，講義内でテストベンチの作り方について説明が少なかったように感じられた．せめて，1 命令のテスト分，また，$display$ 命令の使い方については説明してくれたほうがいいのではないかと考える．今回の $MIPS$ は教科書に従って SE/UE や，MUX も全て $assign$ 文で実装した．そのため，流れが追いやすく，デバッグしやすかった反面，他にも実装パターンは複数考えられるので，本当はさらに効率のいい実装方法があるのだろうなと思った．それとも，コンパイルして論理合成すると，結局同じ回路になるのかどうか興味が湧いた．

4 感想

4.1 森岡悠人

この講義で初めて Verilog や FPGA 開発を経験したが，これまでプログラミング言語しか触れてこなかったため，verilog の独自の命令や module の考え方，並列で回路の処理が実行される考え方は新鮮で面白かった．実際に FPGA が動作したときは達成感を感じた．一方，初めての HDL で，面食らったことも多々あった．modelsim および quartus prime の操作やプロジェクトファイルの構造が独特で，かつ頻繁に強制終了するため，実装に非常に時間がかかった．シミュレーションで，メモリの中身を見る機能や，回路の wave を確認する画面で，自由にソースコードの信号を追加して確認できる機能は非常に便利だった．だが，門屋に教えてもらうまではその存在に気が付かなかったため，もうちょっとわかりやすい UI にしてほしいと思った．ハードウェアを設計する技術者の使うツールだから，そのために用いるソフトウェアもあまり出来が良くないのではないかと思った．また，私が普段利用する ChatGPT のような AI は，おそらく Verilog のコード生成に必要な学習データが不足していて，間違ったコードを生成することが多く，あまり役に立たなかった．また，1 時間計から MIPS に取り掛かるときにレベルが一気に 3 段階くらい上がるなという印象を受けた．

私は将来は低レベルのコンピュータの動作を理解したエンジニアを目指しているため，この講義を受講した．その結果，かつて苦しんだアセンブラの実験の内容をおさらいできたことに加え，FPGA 実機を用いた verilog の回路設計の経験を積むことができ，新たな知見を得られたため，受講して良かったと感じている．また，ペアの松本の実装が速く，常に私に先行して ALU や命令実行時の動作について教えてくれた．起こった問題に対して二人でアイデアを出し合いながら原因を検討し，試行錯誤の中で回路の理解を深められたことが，今回の課題がうまく行った理由ではないかと感じている．

謝辞

本課題に取り組む過程で，忙しい中何度も有益な助言をくださった大崎綾斗さん，門屋陽丈さんに感謝いたします．

付録 A　テストプログラム

本節では、各テストプログラムのアセンブリコードとその動作、および実行結果について説明する。

A.1　load_store

A.1.1　プログラム

このプログラムは基本的なロード・ストア命令の動作を確認するためのテストである。プログラムでは、データメモリから値をロードし、別のアドレスにストアする処理を行う。

```
.set noreorder

.global _start
_start:
lw $s0, 0($s7)
lw $s1, 4($s7)
addi $t0, $s7, 8
sw $s1, 0($t0)
loop:
j loop

.data
a: .word 10
b: .word 20
array1: .space 16
```

プログラムの動作：

- `lw $s0, 0($s7)`:　データメモリのアドレス`$s7+0` から値をロードし、`$s0` に格納
- `lw $s1, 4($s7)`:　データメモリのアドレス`$s7+4` から値をロードし、`$s1` に格納
- `addi $t0, $s7, 8`:　`$s7+8` をアドレスとして`$t0` に計算
- `sw $s1, 0($t0)`:　`$s1` の値を`$t0` が示すアドレスにストア

A.1.2　テスト結果

```
run -all
# Time PC Instruction ALU_Result
# 0 ns 0x00000000 0b00100000111001110000000000000000 0x00000000
# 25000 ns 0x00000004 0b10001110111100000000000000000000 0x00000000
# 35000 ns 0x00000008 0b10001110111100010000000000000100 0x00000001
# 45000 ns 0x0000000c 0b00100010111010000000000000001000 0x00000008
```

```
# 55000 ns 0x00000010 0b10101101000100010000000000000000 0x00000002
# 65000 ns 0x00000014 0b00001000000000000000000000000101 0xxxxxxxxx
# ==== Simulation Results ====
# $s0 register (R16) final value: 0x0000000a
# $s1 register (R17) final value: 0x00000014
# DMem[0] final value: 0x0000000a
# Simulation finished.
```

A.2 arithmetic

A.2.1 プログラム

このプログラムは基本的な算術演算命令の動作を確認するためのテストである。3 つの値を読み込んで加算を行い、結果をレジスタに格納する。

```
.set noreorder

.global _start
_start:
lw $s0, 0($s7)
lw $s1, 4($s7)
lw $s2, 8($s7)
add $t0, $s0, $s1
add $s3, $t0, $s2
loop:
j loop

.data
a: .word 1
b: .word 2
c: .word 3
d: .word 0
```

プログラムの動作：

- 変数 a，b，c の値（1，2，3）をそれぞれ$s0，$s1，$s2 にロード
- add $t0，$s0，$s1:　a + b の結果を$t0 に格納
- add $s3，$t0，$s2:　（a + b）+ c の結果を$s3 に格納（期待値：6）

A.2.2 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.
```

A.3 array

A.3.1 プログラム

このプログラムは配列のアクセスとインデックス計算の動作を確認するためのテストである。

```
.set noreorder

.global _start
_start:
sll $t0, $s0, 2
add $t0, $s7, $t0
lw $s1, 0($t0)
lw $s2, 20($s7)
loop:
j loop

.data
a: .word 10
b: .word 20
array1: .word 0,1,2,3,4,5,6,7,8,9
```

プログラムの動作：

- `sll $t0, $s0, 2`: インデックス`$s0`を 4 倍（左シフト 2 ビット）してワードアドレスに変換
- `add $t0, $s7, $t0`: 配列の基底アドレス`$s7`にオフセットを加算
- `lw $s1, 0($t0)`: 計算されたアドレスから配列要素をロード
- `lw $s2, 20($s7)`: 配列の 5 番目の要素（20 バイトオフセット）をロード

A.3.2 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.
```

A.4 if_then_else

A.4.1 プログラム

このプログラムは条件分岐命令の動作を確認するためのテストである。

```
.set noreorder

.global _start
_start:
beq $s0, $s1, L1
add $s2, $zero, $s0
j N1
L1: add $s2, $zero, $s1
N1:
loop:
j loop

.data
a: .word 10
b: .word 20
array1: .word 0,1,2,3,4,5,6,7,8,9
```

プログラムの動作：

- `beq $s0, $s1, L1:` $s0 と$s1 が等しい場合は L1 にジャンプ
- 等しくない場合：`$s2 = $s0` を実行して N1 にジャンプ

- 等しい場合（L1）：$s2 = $s1 を実行

A.4.2 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.
```

A.4.3 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.
```

A.5 while

A.5.1 プログラム

このプログラムはループ処理と条件判定の動作を確認するためのテストである。

```
.set noreorder

.global _start
_start:
L1: slti $t0, $s0, 10
beq $t0, $zero, N1
sll $t0, $s0, 2
```

```
add $t0, $s7, $t0
sw $s0, 0($t0)
addi $s0, $s0, 1
j L1
N1:
loop:
j loop

.data
a: .word 10
b: .word 20
array1: .space 40
```

プログラムの動作：

- `slti $t0, $s0, 10:  $s0 < 10` **の条件判定結果を**`$t0` **に格納**
- **条件が偽（**`$s0 >= 10`**）の場合はループを終了して** `N1` **にジャンプ**
- **条件が真の場合：配列に**`$s0` **の値を格納し、**`$s0` **をインクリメントしてループを継続**

A.5.2 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 0b00000000000010000010000010000000 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 0b00000000000010000010000010000000 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 0b00000000000010000010000010000000 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 0b00000000000010000010000010000000 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 0b00000000000010000010000010000000 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 0b00000000000010000010000010000000 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 0b00000000000010000010000010000000 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 0b00000000000010000010000010000000 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 ====
# $s0 register (R16) final value: 0x0000000a
# DMem[0] final value: 0x00000000
# DMem[1] final value: 0x00000001
# DMem[2] final value: 0x00000002
# DMem[3] final value: 0x00000003
# DMem[4] final value: 0x00000004
# DMem[5] final value: 0x00000005
# DMem[6] final value: 0x00000006
# DMem[7] final value: 0x00000007
# DMem[8] final value: 0x00000008
# DMem[9] final value: 0x00000009
# Simulation finished.
```

A.6 function

A.6.1 プログラム

このプログラムは関数呼び出しとスタック操作の動作を確認するためのテストである。1 から n までの和を計算する関数を実装している。

```
.set noreorder

.global _start
_start:
```

```
  add $a0, $zero, $s0
  jal sum
  add $s1, $zero, $v0
loop:
j loop
sum:addi $sp, $sp, -8
  sw $s0, 0($sp)
  sw $s1, 4($sp)
  add $s1, $zero, $zero
  add $s0, $zero, $zero
L1: slt $t0, $s0, $a0
  beq $t0, $zero, N1
  add $s1, $s1, $s0
  addi $s1, $s1, 1
  addi $s0, $s0, 1
  j L1
N1:add $v0, $zero, $s1
  lw $s1, 4($sp)
  lw $s0, 0($sp)
  addi $sp, $sp, 8
  jr $ra

.data
a: .word 10
b: .word 20
array1: .space 40
```

プログラムの動作：

- **メイン部分：引数を**$a0 **に設定して** sum **関数を呼び出し、戻り値を**$s1 **に格納**
- sum **関数：スタックにレジスタを退避し、**1 **から** n **までの和を計算して**$v0 **に結果を返す**
- **関数終了時にはスタックからレジスタを復元し、呼び出し元に戻る**

A.6.2 テスト結果

```
run -all
# Time PC Instruction ALU_Result
# 0 ns 0x00000000 0b00000000000100000010000000100000 0x0000000a
# 25000 ns 0x00000004 0b00001100000000000000000000000100 0xxxxxxxxx
# 35000 ns 0x00000010 0b00100011101111011111111111111000 0xfffffff8
```

```
# 45000 ns 0x00000014 0b10101111101100000000000000000000 0xfffffffe
# 55000 ns 0x00000018 0b10101111101100010000000000000100 0xffffffff
# 65000 ns 0x0000001c 0b00000000000000000100010000100000 0x00000000
# 75000 ns 0x00000020 0b00000000000000000100000000100000 0x00000000
# 85000 ns 0x00000024 0b00000010000001000100000000101010 0x00000001
# 95000 ns 0x00000028 0b00010001000000000000000000000100 0x00000001
# 105000 ns 0x0000002c 0b00000010001100001000100000100000 0x00000000
# 115000 ns 0x00000030 0b00100010001100010000000000000001 0x00000001
# 125000 ns 0x00000034 0b00100010000100000000000000000001 0x00000001
# 135000 ns 0x00000038 0b00001000000000000000000000001001 0xxxxxxxxx
# 145000 ns 0x00000024 0b00000010000001000100000000101010 0x00000001
# 155000 ns 0x00000028 0b00010001000000000000000000000100 0x00000001
# 165000 ns 0x0000002c 0b00000010001100001000100000100000 0x00000002
# 175000 ns 0x00000030 0b00100010001100010000000000000001 0x00000003
# 185000 ns 0x00000034 0b00100010000100000000000000000001 0x00000002
# 195000 ns 0x00000038 0b00001000000000000000000000001001 0xxxxxxxxx
# 205000 ns 0x00000024 0b00000010000001000100000000101010 0x00000001
# 215000 ns 0x00000028 0b00010001000000000000000000000100 0x00000001
# 225000 ns 0x0000002c 0b00000010001100001000100000100000 0x00000005
# 235000 ns 0x00000030 0b00100010001100010000000000000001 0x00000006
# 245000 ns 0x00000034 0b00100010000100000000000000000001 0x00000003
# 255000 ns 0x00000038 0b00001000000000000000000000001001 0xxxxxxxxx
# 265000 ns 0x00000024 0b00000010000001000100000000101010 0x00000001
# 275000 ns 0x00000028 0b00010001000000000000000000000100 0x00000001
# 285000 ns 0x0000002c 0b00000010001100001000100000100000 0x00000009
# 295000 ns 0x00000030 0b00100010001100010000000000000001 0x0000000a
# 305000 ns 0x00000034 0b00100010000100000000000000000001 0x00000004
# 315000 ns 0x00000038 0b00001000000000000000000000001001 0xxxxxxxxx
# 325000 ns 0x00000024 0b00000010000001000100000000101010 0x00000001
# 335000 ns 0x00000028 0b00010001000000000000000000000100 0x00000001
# 345000 ns 0x0000002c 0b00000010001100001000100000100000 0x0000000e
# 355000 ns 0x00000030 0b00100010001100010000000000000001 0x0000000f
# 365000 ns 0x00000034 0b00100010000100000000000000000001 0x00000005
# 375000 ns 0x00000038 0b00001000000000000000000000001001 0xxxxxxxxx
# 385000 ns 0x00000024 0b00000010000001000100000000101010 0x00000001
# 395000 ns 0x00000028 0b00010001000000000000000000000100 0x00000001
# 405000 ns 0x0000002c 0b00000010001100001000100000100000 0x00000014
# 415000 ns 0x00000030 0b00100010001100010000000000000001 0x00000015
# 425000 ns 0x00000034 0b00100010000100000000000000000001 0x00000006
# 435000 ns 0x00000038 0b00001000000000000000000000001001 0xxxxxxxxx
```

```
# 445000 ns 0x00000024 0b00000010000001000100000000101010 0x00000001
# 455000 ns 0x00000028 0b00010001000000000000000000000100 0x00000001
# 465000 ns 0x0000002c 0b00000010001100001000100000100000 0x0000001b
# 475000 ns 0x00000030 0b00100010001100010000000000000001 0x0000001c
# 485000 ns 0x00000034 0b00100010000100000000000000000001 0x00000007
# 495000 ns 0x00000038 0b00001000000000000000000000001001 0xxxxxxxxx
# 505000 ns 0x00000024 0b00000010000001000100000000101010 0x00000001
# 515000 ns 0x00000028 0b00010001000000000000000000000100 0x00000001
# 525000 ns 0x0000002c 0b00000010001100001000100000100000 0x00000023
# 535000 ns 0x00000030 0b00100010001100010000000000000001 0x00000024
# 545000 ns 0x00000034 0b00100010000100000000000000000001 0x00000008
# 555000 ns 0x00000038 0b00001000000000000000000000001001 0xxxxxxxxx
# 565000 ns 0x00000024 0b00000010000001000100000000101010 0x00000001
# 575000 ns 0x00000028 0b00010001000000000000000000000100 0x00000001
# 585000 ns 0x0000002c 0b00000010001100001000100000100000 0x0000002c
# 595000 ns 0x00000030 0b00100010001100010000000000000001 0x0000002d
# 605000 ns 0x00000034 0b00100010000100000000000000000001 0x00000009
# 615000 ns 0x00000038 0b00001000000000000000000000001001 0xxxxxxxxx
# 625000 ns 0x00000024 0b00000010000001000100000000101010 0x00000001
# 635000 ns 0x00000028 0b00010001000000000000000000000100 0x00000001
# 645000 ns 0x0000002c 0b00000010001100001000100000100000 0x00000036
# 655000 ns 0x00000030 0b00100010001100010000000000000001 0x00000037
# 665000 ns 0x00000034 0b00100010000100000000000000000001 0x0000000a
# 675000 ns 0x00000038 0b00001000000000000000000000001001 0xxxxxxxxx
# 685000 ns 0x00000024 0b00000010000001000100000000101010 0x00000000
# 695000 ns 0x00000028 0b00010001000000000000000000000100 0x00000000
# 705000 ns 0x0000003c 0b00000000000100010001000000100000 0x00000037
# 715000 ns 0x00000040 0b10001111101100010000000000000100 0xffffffff
# 725000 ns 0x00000044 0b10001111101100000000000000000000 0xfffffffe
# 735000 ns 0x00000048 0b00100011101111010000000000001000 0x00000000
# 745000 ns 0x0000004c 0b00000011111000000000000000001000 0xxxxxxxxx
# 755000 ns 0x00000008 0b00000000000000101000100000100000 0x00000037
# 765000 ns 0x0000000c 0b00001000000000000000000000000011 0xxxxxxxxx
# ==== Simulation Results ====
# $s1 register (R17) final value: 0x00000037
# Simulation finished.
```

A.7 recursion

A.7.1 プログラム

このプログラムは再帰関数呼び出しの動作を確認するためのテストである。再帰を用いて 1 から n までの和を計算する。

```
.set noreorder

.global _start
_start:
  add $a0, $zero, $s0
  jal sum
  add $s1, $zero, $v0
loop:
j loop
sum:addi $sp, $sp, -8
  sw $a0, 0($sp)
  sw $ra, 4($sp)
  slti $t0, $a0, 1  # Check if a0 < 1
  beq $t0, $zero, L1  # If a0 >= 1, go to L1
  lw $ra, 4($sp)
  lw $a0, 0($sp)
  add $v0, $zero, $zero
  addi $sp, $sp, 8
  jr $ra
L1: addi $a0, $a0, -1
  jal sum
  lw $a0, 0($sp)
  lw $ra, 4($sp)
  add $v0, $v0, $a0
  addi $sp, $sp, 8
  jr $ra

.data
a: .word 10
b: .word 20
array1: .space 40
```

プログラムの動作：

- 引数が 1 未満の場合は 0 を返してベースケースとする
- 引数が 1 以上の場合は、引数を 1 減らして再帰呼び出しを行い、その結果に現在の引数値を加算
- 各再帰レベルでスタックに引数と戻りアドレスを保存・復元

A.7.2 テスト結果

```
run -all
# Time PC Instruction ALU_Result
# 0 ns 0x00000000 0b00000000000010000001000000100000 0x0000000a
# 25000 ns 0x00000004 0b00001100000000000000000000000100 0xxxxxxxxx
# 35000 ns 0x00000010 0b00100011101111011111111111111000 0xfffffff8
# 45000 ns 0x00000014 0b10101111101001000000000000000000 0xfffffffe
# 55000 ns 0x00000018 0b10101111101111110000000000000100 0xffffffff
# 65000 ns 0x0000001c 0b00101000100010000000000000000001 0x00000000
# 75000 ns 0x00000020 0b00010001000000000000000000000101 0x00000000
# 85000 ns 0x00000038 0b00100000100001001111111111111111 0x00000009
# 95000 ns 0x0000003c 0b00001100000000000000000000000100 0xxxxxxxxx
# 105000 ns 0x00000010 0b00100011101111011111111111111000 0xfffffff0
# 115000 ns 0x00000014 0b10101111101001000000000000000000 0xfffffffc
# 125000 ns 0x00000018 0b10101111101111110000000000000100 0xfffffffd
# 135000 ns 0x0000001c 0b00101000100010000000000000000001 0x00000000
# 145000 ns 0x00000020 0b00010001000000000000000000000101 0x00000000
# 155000 ns 0x00000038 0b00100000100001001111111111111111 0x00000008
# 165000 ns 0x0000003c 0b00001100000000000000000000000100 0xxxxxxxxx
# 175000 ns 0x00000010 0b00100011101111011111111111111000 0xfffffff8
# 185000 ns 0x00000014 0b10101111101001000000000000000000 0xfffffffa
# 195000 ns 0x00000018 0b10101111101111110000000000000100 0xfffffffb
# 205000 ns 0x0000001c 0b00101000100010000000000000000001 0x00000000
# 215000 ns 0x00000020 0b00010001000000000000000000000101 0x00000000
# 225000 ns 0x00000038 0b00100000100001001111111111111111 0x00000007
# 235000 ns 0x0000003c 0b00001100000000000000000000000100 0xxxxxxxxx
# 245000 ns 0x00000010 0b00100011101111011111111111111000 0xfffffff0
# 255000 ns 0x00000014 0b10101111101001000000000000000000 0xfffffff8
# 265000 ns 0x00000018 0b10101111101111110000000000000100 0xfffffff9
# 275000 ns 0x0000001c 0b00101000100010000000000000000001 0x00000000
# 285000 ns 0x00000020 0b00010001000000000000000000000101 0x00000000
# 295000 ns 0x00000038 0b00100000100001001111111111111111 0x00000006
# 305000 ns 0x0000003c 0b00001100000000000000000000000100 0xxxxxxxxx
# 315000 ns 0x00000010 0b00100011101111011111111111111000 0xfffffff8
```

```
# 325000 ns 0x00000014 0b10101111101001000000000000000000 0xfffffff6
# 335000 ns 0x00000018 0b10101111101111110000000000000100 0xfffffff7
# 345000 ns 0x0000001c 0b00101000100010000000000000000001 0x00000000
# 355000 ns 0x00000020 0b00010001000000000000000000000101 0x00000000
# 365000 ns 0x00000038 0b00100000100001001111111111111111 0x00000005
# 375000 ns 0x0000003c 0b00001100000000000000000000000100 0xxxxxxxxx
# 385000 ns 0x00000010 0b00100011101111011111111111111000 0xfffffff0
# 395000 ns 0x00000014 0b10101111101001000000000000000000 0xfffffff4
# 405000 ns 0x00000018 0b10101111101111110000000000000100 0xfffffff5
# 415000 ns 0x0000001c 0b00101000100010000000000000000001 0x00000000
# 425000 ns 0x00000020 0b00010001000000000000000000000101 0x00000000
# 435000 ns 0x00000038 0b00100000100001001111111111111111 0x00000004
# 445000 ns 0x0000003c 0b00001100000000000000000000000100 0xxxxxxxxx
# 455000 ns 0x00000010 0b00100011101111011111111111111000 0xfffffff8
# 465000 ns 0x00000014 0b10101111101001000000000000000000 0xfffffff2
# 475000 ns 0x00000018 0b10101111101111110000000000000100 0xfffffff3
# 485000 ns 0x0000001c 0b00101000100010000000000000000001 0x00000000
# 495000 ns 0x00000020 0b00010001000000000000000000000101 0x00000000
# 505000 ns 0x00000038 0b00100000100001001111111111111111 0x00000003
# 515000 ns 0x0000003c 0b00001100000000000000000000000100 0xxxxxxxxx
# 525000 ns 0x00000010 0b00100011101111011111111111111000 0xfffffff0
# 535000 ns 0x00000014 0b10101111101001000000000000000000 0xfffffff0
# 545000 ns 0x00000018 0b10101111101111110000000000000100 0xfffffff1
# 555000 ns 0x0000001c 0b00101000100010000000000000000001 0x00000000
# 565000 ns 0x00000020 0b00010001000000000000000000000101 0x00000000
# 575000 ns 0x00000038 0b00100000100001001111111111111111 0x00000002
# 585000 ns 0x0000003c 0b00001100000000000000000000000100 0xxxxxxxxx
# 595000 ns 0x00000010 0b00100011101111011111111111111000 0xfffffff8
# 605000 ns 0x00000014 0b10101111101001000000000000000000 0xfffffffe
# 615000 ns 0x00000018 0b10101111101111110000000000000100 0xfffffffe
# 625000 ns 0x0000001c 0b00101000100010000000000000000001 0x00000000
# 635000 ns 0x00000020 0b00010001000000000000000000000101 0x00000000
# 645000 ns 0x00000038 0b00100000100001001111111111111111 0x00000001
# 655000 ns 0x0000003c 0b00001100000000000000000000000100 0xxxxxxxxx
# 665000 ns 0x00000010 0b00100011101111011111111111111000 0xfffffff0
# 675000 ns 0x00000014 0b10101111101001000000000000000000 0xfffffffe
# 685000 ns 0x00000018 0b10101111101111110000000000000100 0xfffffffe
# 695000 ns 0x0000001c 0b00101000100010000000000000000001 0x00000000
# 705000 ns 0x00000020 0b00010001000000000000000000000101 0x00000000
# 715000 ns 0x00000038 0b00100000100001001111111111111111 0x00000000
```

```
# 725000 ns 0x0000003c 0b00001100000000000000000000000100 0xxxxxxxx
# 735000 ns 0x00000010 0b00100011101111011111111111111000 0xfffffa8
# 745000 ns 0x00000014 0b10101111101001000000000000000000 0xfffffea
# 755000 ns 0x00000018 0b10101111101111110000000000000100 0xfffffeb
# 765000 ns 0x0000001c 0b00101000100010000000000000000001 0x00000001
# 775000 ns 0x00000020 0b00010001000000000000000000000101 0x00000001
# 785000 ns 0x00000024 0b10001111101111110000000000000100 0xfffffeb
# 795000 ns 0x00000028 0b10001111101001000000000000000000 0xfffffea
# 805000 ns 0x0000002c 0b00000000000000000001000000100000 0x00000000
# 815000 ns 0x00000030 0b00100011101111010000000000001000 0xfffffb0
# 825000 ns 0x00000034 0b00000011111000000000000000001000 0xxxxxxxxx
# 835000 ns 0x00000040 0b10001111101001000000000000000000 0xfffffec
# 845000 ns 0x00000044 0b10001111101111110000000000000100 0xfffffed
# 855000 ns 0x00000048 0b00000000010001000001000000100000 0x00000001
# 865000 ns 0x0000004c 0b00100011101111010000000000001000 0xfffffb8
# 875000 ns 0x00000050 0b00000011111000000000000000001000 0xxxxxxxxx
# 885000 ns 0x00000040 0b10001111101001000000000000000000 0xfffffee
# 895000 ns 0x00000044 0b10001111101111110000000000000100 0xfffffef
# 905000 ns 0x00000048 0b00000000010001000001000000100000 0x00000003
# 915000 ns 0x0000004c 0b00100011101111010000000000001000 0xfffffc0
# 925000 ns 0x00000050 0b00000011111000000000000000001000 0xxxxxxxxx
# 935000 ns 0x00000040 0b10001111101001000000000000000000 0xfffffff0
# 945000 ns 0x00000044 0b10001111101111110000000000000100 0xfffffff1
# 955000 ns 0x00000048 0b00000000010001000001000000100000 0x00000006
# 965000 ns 0x0000004c 0b00100011101111010000000000001000 0xfffffc8
# 975000 ns 0x00000050 0b00000011111000000000000000001000 0xxxxxxxxx
# 985000 ns 0x00000040 0b10001111101001000000000000000000 0xfffffff2
# 995000 ns 0x00000044 0b10001111101111110000000000000100 0xfffffff3
# 1005000 ns 0x00000048 0b00000000010001000001000000100000 0x0000000a
# 1015000 ns 0x0000004c 0b00100011101111010000000000001000 0xfffffd0
# 1025000 ns 0x00000050 0b00000011111000000000000000001000 0xxxxxxxxx
# 1035000 ns 0x00000040 0b10001111101001000000000000000000 0xfffffff4
# 1045000 ns 0x00000044 0b10001111101111110000000000000100 0xfffffff5
# 1055000 ns 0x00000048 0b00000000010001000001000000100000 0x0000000f
# 1065000 ns 0x0000004c 0b00100011101111010000000000001000 0xfffffd8
# 1075000 ns 0x00000050 0b00000011111000000000000000001000 0xxxxxxxxx
# 1085000 ns 0x00000040 0b10001111101001000000000000000000 0xfffffff6
# 1095000 ns 0x00000044 0b10001111101111110000000000000100 0xfffffff7
# 1105000 ns 0x00000048 0b00000000010001000001000000100000 0x00000015
# 1115000 ns 0x0000004c 0b00100011101111010000000000001000 0xfffffe0
```

```
# 1125000 ns 0x00000050 0b00000011111000000000000000001000 0xxxxxxxxx
# 1135000 ns 0x00000040 0b10001111101001000000000000000000 0xfffffff8
# 1145000 ns 0x00000044 0b10001111101111110000000000000100 0xfffffff9
# 1155000 ns 0x00000048 0b00000000010001000001000000100000 0x0000001c
# 1165000 ns 0x0000004c 0b00100011101111010000000000001000 0xffffffe8
# 1175000 ns 0x00000050 0b00000011111000000000000000001000 0xxxxxxxxx
# 1185000 ns 0x00000040 0b10001111101001000000000000000000 0xfffffffa
# 1195000 ns 0x00000044 0b10001111101111110000000000000100 0xfffffffb
# 1205000 ns 0x00000048 0b00000000010001000001000000100000 0x00000024
# 1215000 ns 0x0000004c 0b00100011101111010000000000001000 0xfffffff0
# 1225000 ns 0x00000050 0b00000011111000000000000000001000 0xxxxxxxxx
# 1235000 ns 0x00000040 0b10001111101001000000000000000000 0xfffffffc
# 1245000 ns 0x00000044 0b10001111101111110000000000000100 0xfffffffd
# 1255000 ns 0x00000048 0b00000000010001000001000000100000 0x0000002d
# 1265000 ns 0x0000004c 0b00100011101111010000000000001000 0xfffffff8
# 1275000 ns 0x00000050 0b00000011111000000000000000001000 0xxxxxxxxx
# 1285000 ns 0x00000040 0b10001111101001000000000000000000 0xfffffffe
# 1295000 ns 0x00000044 0b10001111101111110000000000000100 0xffffffff
# 1305000 ns 0x00000048 0b00000000010001000001000000100000 0x00000037
# 1315000 ns 0x0000004c 0b00100011101111010000000000001000 0x00000000
# 1325000 ns 0x00000050 0b00000011111000000000000000001000 0xxxxxxxxx
# 1335000 ns 0x00000008 0b00000000000000101000100000100000 0x00000037
# 1345000 ns 0x0000000c 0b00001000000000000000000000000011 0xxxxxxxxx
# ==== Simulation Results ====
# $s1 register (R17) final value: 0x00000037
# Simulation finished.
```

A.8 hanoi

A.8.1 プログラム

このプログラムはハノイの塔を解く再帰アルゴリズムを実装し、複雑な再帰処理の動作を確認するためのテストである。

```
.set noreorder

.global _start
_start:
    # init
    addi $a0, $zero, 3 # $a0 = $zero + 3, n
    addi $a1, $zero, 3 # $a1 = $zero + 1, from
```

```
    addi $a2, $zero, 0 # $a2 = $zero + 2, to
    addi    $a3, $zero, 0           # $a3 = $zero + 0, aux
    addi    $t1, $zero, 0           # for count
    # call
    jal hanoi # jump to hanoi and save position to $ra

loop:
    j loop # jump to loop
hanoi:
    addi $sp, $sp, -20 # $sp = $sp + -20
    sw $a0, 0($sp)
    sw      $a1, 4($sp)
    sw      $a2, 8($sp)
    sw      $a3, 12($sp)
    sw      $ra, 16($sp)
    slti $t0, $a0, 2 # $t0 = ($a0 < 2) ? 1 : 0
    beq $t0, $zero, L1     # if $t0 == $zero then goto L1
    add $a2, $a3, $zero # $a2 = $a3 + $zero
    addi    $t1, $t1, 1           # $t1 = $t1 + 1
    lw $a2, 8($sp)
    lw      $ra, 16($sp)
    addi $sp, $sp, 20 # $sp = $sp + 20
    jr $ra # jump to $ra
L1:
    addi $a0, $a0, -1 # $a0 = $a0 + -1
    lw      $a2, 12($sp)
    lw      $a3, 8($sp)
    jal hanoi # jump to hanoi and save position to $ra
    lw      $a0, 0($sp)
    addi    $t1, $t1, 1           # $t1 = $t1 + 1
    addi $a0, $a0, -1 # $a0 = $a0 + -1
    lw      $a1, 8($sp)
    lw      $a2, 4($sp)
    lw      $a3, 12($sp)
    jal hanoi # jump to hanoi and save position to $ra
    lw      $a0, 0($sp)
    lw      $a1, 4($sp)
    lw      $a2, 8($sp)
    lw      $a3, 12($sp)
    lw      $ra, 16($sp)
```

```
addi $sp, $sp, 20 # $sp = $sp + 20
jr $ra # jump to $ra
```

プログラムの動作：

- 3 枚の円盤のハノイの塔問題を解く
- $a0: 円盤の枚数、$a1: 移動元、$a2: 移動先、$a3: 補助杆
- $t1: 移動回数カウンタ（期待値：7 回）
- 再帰の深さに応じてスタックに多くの引数と戻りアドレスを保存
- ベースケース（円盤が 1 枚）では直接移動を実行

A.8.2 テスト結果

```
run -all
# Time PC Instruction ALU_Result
# 0 ns 0x00000000 0b00100000000001000000000000000011 0x00000003
# 25000 ns 0x00000004 0b00100000000001010000000000000011 0x00000003
# 35000 ns 0x00000008 0b00100000000001100000000000000000 0x00000000
# 45000 ns 0x0000000c 0b00100000000001110000000000000000 0x00000000
# 55000 ns 0x00000010 0b00100000000010010000000000000000 0x00000000
# 65000 ns 0x00000014 0b00000110000000000000000000000111 0xxxxxxxxx
# 75000 ns 0x0000001c 0b00100011101111011111111111101100 0xfffffec
# 85000 ns 0x00000020 0b10101111101001000000000000000000 0xfffffffb
# 95000 ns 0x00000024 0b10101111101001010000000000000100 0xfffffffc
# 105000 ns 0x00000028 0b10101111101001100000000000001000 0xfffffffd
# 115000 ns 0x0000002c 0b10101111101001110000000000001100 0xfffffffe
# 125000 ns 0x00000030 0b10101111101111110000000000010000 0xffffffff
# 135000 ns 0x00000034 0b00101000100010000000000000000010 0x00000000
# 145000 ns 0x00000038 0b00010001000000000000000000000110 0x00000000
# 155000 ns 0x00000054 0b00100000100001001111111111111111 0x00000002
# 165000 ns 0x00000058 0b10001111101001100000000000001100 0xfffffffe
# 175000 ns 0x0000005c 0b10001111101001110000000000001000 0xfffffffd
# 185000 ns 0x00000060 0b00000110000000000000000000000111 0xxxxxxxxx
# 195000 ns 0x0000001c 0b00100011101111011111111111101100 0xfffffffd8
# 205000 ns 0x00000020 0b10101111101001000000000000000000 0xfffffff6
# 215000 ns 0x00000024 0b10101111101001010000000000000100 0xfffffff7
# 225000 ns 0x00000028 0b10101111101001100000000000001000 0xfffffff8
# 235000 ns 0x0000002c 0b10101111101001110000000000001100 0xfffffff9
# 245000 ns 0x00000030 0b10101111101111110000000000010000 0xfffffffa
# 255000 ns 0x00000034 0b00101000100010000000000000000010 0x00000000
# 265000 ns 0x00000038 0b00010001000000000000000000000110 0x00000000
```

```
# 275000 ns 0x00000054 0b00100000100001001111111111111111 0x00000001
# 285000 ns 0x00000058 0b10001111101001100000000000001100 0xfffffff9
# 295000 ns 0x0000005c 0b10001111101001110000000000001000 0xfffffff8
# 305000 ns 0x00000060 0b00001100000000000000000000000111 0xxxxxxxxx
# 315000 ns 0x0000001c 0b00100011101111011111111111101100 0xfffffc4
# 325000 ns 0x00000020 0b10101111101001000000000000000000 0xfffffff1
# 335000 ns 0x00000024 0b10101111101001010000000000000100 0xfffffff2
# 345000 ns 0x00000028 0b10101111101001100000000000001000 0xfffffff3
# 355000 ns 0x0000002c 0b10101111101001110000000000001100 0xfffffff4
# 365000 ns 0x00000030 0b10101111101111110000000000010000 0xfffffff5
# 375000 ns 0x00000034 0b00101000100010000000000000000010 0x00000001
# 385000 ns 0x00000038 0b00010001000000000000000000000110 0x00000001
# 395000 ns 0x0000003c 0b00000000111000000011000000100000 0x00000000
# 405000 ns 0x00000040 0b00100001001010010000000000000001 0x00000001
# 415000 ns 0x00000044 0b10001111101001100000000000001000 0xfffffff3
# 425000 ns 0x00000048 0b10001111101111110000000000010000 0xfffffff5
# 435000 ns 0x0000004c 0b00100011101111010000000000010100 0xfffffffd8
# 445000 ns 0x00000050 0b00000011111000000000000000001000 0xxxxxxxxx
# 455000 ns 0x00000064 0b10001111101001000000000000000000 0xfffffff6
# 465000 ns 0x00000068 0b00100001001010010000000000000001 0x00000002
# 475000 ns 0x0000006c 0b00100000100001001111111111111111 0x00000001
# 485000 ns 0x00000070 0b10001111101001010000000000001000 0xfffffff8
# 495000 ns 0x00000074 0b10001111101001100000000000000100 0xfffffff7
# 505000 ns 0x00000078 0b10001111101001110000000000001100 0xfffffff9
# 515000 ns 0x0000007c 0b00001100000000000000000000000111 0xxxxxxxxx
# 525000 ns 0x0000001c 0b00100011101111011111111111101100 0xfffffffc4
# 535000 ns 0x00000020 0b10101111101001000000000000000000 0xfffffff1
# 545000 ns 0x00000024 0b10101111101001010000000000000100 0xfffffff2
# 555000 ns 0x00000028 0b10101111101001100000000000001000 0xfffffff3
# 565000 ns 0x0000002c 0b10101111101001110000000000001100 0xfffffff4
# 575000 ns 0x00000030 0b10101111101111110000000000010000 0xfffffff5
# 585000 ns 0x00000034 0b00101000100010000000000000000010 0x00000001
# 595000 ns 0x00000038 0b00010001000000000000000000000110 0x00000001
# 605000 ns 0x0000003c 0b00000000111000000011000000100000 0x00000000
# 615000 ns 0x00000040 0b00100001001010010000000000000001 0x00000003
# 625000 ns 0x00000044 0b10001111101001100000000000001000 0xfffffff3
# 635000 ns 0x00000048 0b10001111101111110000000000010000 0xfffffff5
# 645000 ns 0x0000004c 0b00100011101111010000000000010100 0xfffffffd8
# 655000 ns 0x00000050 0b00000011111000000000000000001000 0xxxxxxxxx
# 665000 ns 0x00000080 0b10001111101001000000000000000000 0xfffffff6
```

```
# 675000 ns 0x00000084 0b10001111101001010000000000000100 0xfffffff7
# 685000 ns 0x00000088 0b10001111101001100000000000001000 0xfffffff8
# 695000 ns 0x0000008c 0b10001111101001110000000000001100 0xfffffff9
# 705000 ns 0x00000090 0b10001111101111110000000000010000 0xfffffffa
# 715000 ns 0x00000094 0b00100011101111010000000000010100 0xffffffec
# 725000 ns 0x00000098 0b00000011111000000000000000001000 0xxxxxxxxx
# 735000 ns 0x00000064 0b10001111101001000000000000000000 0xfffffffb
# 745000 ns 0x00000068 0b00100001001010010000000000000001 0x00000004
# 755000 ns 0x0000006c 0b00100000100001001111111111111111 0x00000002
# 765000 ns 0x00000070 0b10001111101001010000000000001000 0xfffffffd
# 775000 ns 0x00000074 0b10001111101001100000000000000100 0xfffffffc
# 785000 ns 0x00000078 0b10001111101001110000000000001100 0xfffffffe
# 795000 ns 0x0000007c 0b00001100000000000000000000000111 0xxxxxxxxx
# 805000 ns 0x0000001c 0b00100011101111011111111111101100 0xffffffd8
# 815000 ns 0x00000020 0b10101111101001000000000000000000 0xfffffff6
# 825000 ns 0x00000024 0b10101111101001010000000000000100 0xfffffff7
# 835000 ns 0x00000028 0b10101111101001100000000000001000 0xfffffff8
# 845000 ns 0x0000002c 0b10101111101001110000000000001100 0xfffffff9
# 855000 ns 0x00000030 0b10101111101111110000000000010000 0xfffffffa
# 865000 ns 0x00000034 0b00101000100010000000000000000010 0x00000000
# 875000 ns 0x00000038 0b00010001000000000000000000000110 0x00000000
# 885000 ns 0x00000054 0b00100000100001001111111111111111 0x00000001
# 895000 ns 0x00000058 0b10001111101001100000000000001100 0xfffffff9
# 905000 ns 0x0000005c 0b10001111101001110000000000001000 0xfffffff8
# 915000 ns 0x00000060 0b00001100000000000000000000000111 0xxxxxxxxx
# 925000 ns 0x0000001c 0b00100011101111011111111111101100 0xffffffc4
# 935000 ns 0x00000020 0b10101111101001000000000000000000 0xfffffff1
# 945000 ns 0x00000024 0b10101111101001010000000000000100 0xfffffff2
# 955000 ns 0x00000028 0b10101111101001100000000000001000 0xfffffff3
# 965000 ns 0x0000002c 0b10101111101001110000000000001100 0xfffffff4
# 975000 ns 0x00000030 0b10101111101111110000000000010000 0xfffffff5
# 985000 ns 0x00000034 0b00101000100010000000000000000010 0x00000001
# 995000 ns 0x00000038 0b00010001000000000000000000000110 0x00000001
# 1005000 ns 0x0000003c 0b00000000111000000011000000100000 0x00000003
# 1015000 ns 0x00000040 0b00100001001010010000000000000001 0x00000005
# 1025000 ns 0x00000044 0b10001111101001100000000000001000 0xfffffff3
# 1035000 ns 0x00000048 0b10001111101111110000000000010000 0xfffffff5
# 1045000 ns 0x0000004c 0b00100011101111010000000000010100 0xffffffd8
# 1055000 ns 0x00000050 0b00000011111000000000000000001000 0xxxxxxxxx
# 1065000 ns 0x00000064 0b10001111101001000000000000000000 0xfffffff6
```

```
# 1075000 ns 0x00000068 0b00100001001010010000000000000001 0x00000006
# 1085000 ns 0x0000006c 0b00100001000010011111111111111111 0x00000001
# 1095000 ns 0x00000070 0b10001111101001010000000000001000 0xfffffff8
# 1105000 ns 0x00000074 0b10001111101001100000000000000100 0xfffffff7
# 1115000 ns 0x00000078 0b10001111101001110000000000001100 0xfffffff9
# 1125000 ns 0x0000007c 0b00001100000000000000000000000111 0xxxxxxxxx
# 1135000 ns 0x0000001c 0b00100011101111011111111111101100 0xfffffc4
# 1145000 ns 0x00000020 0b10101111101001000000000000000000 0xfffffff1
# 1155000 ns 0x00000024 0b10101111101001010000000000000100 0xfffffff2
# 1165000 ns 0x00000028 0b10101111101001100000000000001000 0xfffffff3
# 1175000 ns 0x0000002c 0b10101111101001110000000000001100 0xfffffff4
# 1185000 ns 0x00000030 0b10101111101111110000000000010000 0xfffffff5
# 1195000 ns 0x00000034 0b00101000100010000000000000000010 0x00000001
# 1205000 ns 0x00000038 0b00010001000000000000000000000110 0x00000001
# 1215000 ns 0x0000003c 0b00000000111000000011000000100000 0x00000000
# 1225000 ns 0x00000040 0b00100001001010010000000000000001 0x00000007
# 1235000 ns 0x00000044 0b10001111101001100000000000001000 0xfffffff3
# 1245000 ns 0x00000048 0b10001111101111110000000000010000 0xfffffff5
# 1255000 ns 0x0000004c 0b00100011101111010000000000010100 0xfffffd8
# 1265000 ns 0x00000050 0b00000011111000000000000000001000 0xxxxxxxxx
# 1275000 ns 0x00000080 0b10001111101001000000000000000000 0xfffffff6
# 1285000 ns 0x00000084 0b10001111101001010000000000000100 0xfffffff7
# 1295000 ns 0x00000088 0b10001111101001100000000000001000 0xfffffff8
# 1305000 ns 0x0000008c 0b10001111101001110000000000001100 0xfffffff9
# 1315000 ns 0x00000090 0b10001111101111110000000000010000 0xfffffffa
# 1325000 ns 0x00000094 0b00100011101111010000000000010100 0xffffffec
# 1335000 ns 0x00000098 0b00000011111000000000000000001000 0xxxxxxxxx
# 1345000 ns 0x00000080 0b10001111101001000000000000000000 0xfffffffb
# 1355000 ns 0x00000084 0b10001111101001010000000000000100 0xfffffffc
# 1365000 ns 0x00000088 0b10001111101001100000000000001000 0xfffffffd
# 1375000 ns 0x0000008c 0b10001111101001110000000000001100 0xfffffffe
# 1385000 ns 0x00000090 0b10001111101111110000000000010000 0xffffffff
# 1395000 ns 0x00000094 0b00100011101111010000000000010100 0x00000000
# 1405000 ns 0x00000098 0b00000011111000000000000000001000 0xxxxxxxxx
# 1415000 ns 0x00000018 0b00001000000000000000000000000110 0xxxxxxxxx
# ==== Simulation Results ====
# $t1 register (R9) final value: 0x00000007
# Simulation finished.
```

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.