

CPU実験 1班最終レポート

メンバー

- ・ コア係 五反田
- ・ コンパイラ係 松下
- ・ シミュレータ係 毛利
- ・ □係 坂本

ISAについて (担当: 五反田)

主な特徴

- ・ ワードサイズ: 32bit
- ・ ワード単位アドレッシング
 - 結果、シーケンシャルな命令実行時、PCの増加は4ではなく1
- ・ ハーバードアーキテクチャ
 - 命令メモリ: 0x0000 ~ 0x3FFF (2^14 words)
 - データメモリ: 0x00000 ~ 0x3FFFF (2^18 words)

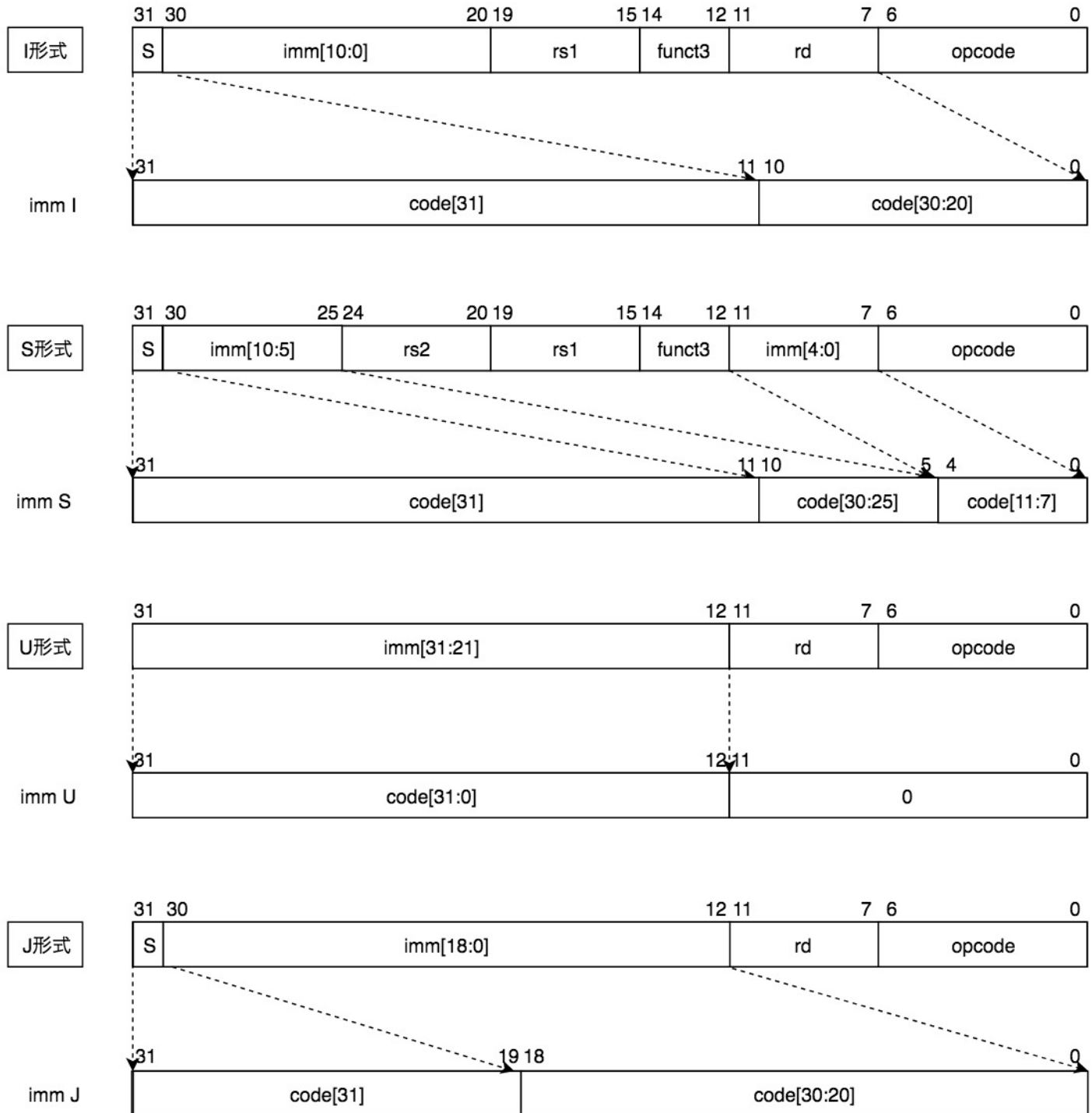
命令および即値のフォーマット

- ・ RISC-Vの命令形式を元に5つの命令フォーマット(R,I,S,U,F)を策定した。
 - そのうち、3形式(I,S,U)は即値を持ち、U形式はさらに即値の形式として、U形式およびJ形式の2形式に分類される。

命令フォーマット

R形式	31 funct7	25 24 rs2	20 19 rs1	15 14 funct3	12 11 rd	7 6 opcode	0
I形式	31 imm[11:0]	20 19 rs1	15 14 funct3	12 11 rd	7 6 opcode		0
S形式	31 imm[11:5]	25 24 rs2	20 19 rs1	15 14 funct3	12 11 imm[4:0]	7 6 opcode	0
U形式	31 imm[31:21] / imm[19:0]			12 11 rd	7 6 opcode		0
F形式	31 funct5	27 26 25 24 fmt	20 19 rs2	15 14 rs1	12 11 rm	7 6 rd	0 opcode

即値フォーマット



レジスタ

- ・ プログラムカウンタ: pc
- ・ 汎用レジスタ: 32個 (x0~x31)
 - x0は0で固定される。
 - x0への書き込みは棄却される。
- ・ 浮動小数点レジスタ: 32個 (f0~f31)
 - f0およびf11~f29は以下に示す値に固定。
 - x0と同様f0,f11~f29への書き込みは棄却される。

レジスタ名	値	bit表現	備考
f0	0.0	0x00000000	
f11	1.0	0x3F800000	//LUIで生成可能
f12	2.0	0x40000000	//LUIで生成可能
f13	4.0	0x40800000	//LUIで生成可能
f14	10.0	0x41200000	//LUIで生成可能
f15	15.0	0x41700000	//LUIで生成可能
f16	20.0	0x41A00000	//LUIで生成可能
f17	128.0	0x43000000	//LUIで生成可能
f18	200.0	0x43480000	//LUIで生成可能
f19	255.0	0x437F0000	//LUIで生成可能
f20	850.0	0x44548000	//LUIで生成可能
f21	0.100	0x3DCCCCCD	//LUI&ADDIで生成可能
f22	0.200	0x3E4CCCCD	//LUI&ADDIで生成可能
f23	0.001	0x3A83126F	//LUI&ADDIで生成可能
f24	0.005	0x3BA3D70A	//LUI&ADDIで生成可能
f25	0.150	0x3E19999A	//LUI&ADDIで生成可能
f26	0.250	0x3E800000	//LUIで生成可能
f27	0.500	0x3F000000	//LUIで生成可能
f28	pi	0x40490FDB	//LUI&ADDIで生成可能
f29	30.0 / pi	0x4118C9EB	//LUI&ADDIで生成可能

基本命令(RV32I改)

命令	opcode	形式	解釈疑似コード	命令(即値)フォーマット
lui	0b0110111	lui rd, imm	rd = imm<<12, pc++	U
auipc	0b0010111	auipc rd, imm	rd = pc + (imm<<12), pc++	U
jal	0b1101111	jal rd, imm	rd = pc + 1, pc += imm	J
jalr	0b1100111	jalr rd, rs1, imm	rd = pc + 1, pc = rs1 + imm	I
beq	0b1100011	beq rs1, rs2, pc + (imm<<2)	if(rs1 == rs2) then pc += imm else pc++	B
bne	同上	bne rs1, rs2, pc + (imm<<2)	if(rs1 != rs2) then pc += imm else pc++	B
blt	同上	blt rs1, rs2, pc + (imm<<2)	if(rs1 < rs2) then pc += imm else pc++	B
bge	同上	bge rs1, rs2, pc + (imm<<2)	if(rs1 >= rs2) then pc += imm else pc++	B
bltu	同上	bltu rs1, rs2, pc + (imm<<2)	if(rs1 < rs2) then pc += imm else pc++	B
bgeu	同上	bgeu rs1, rs2, pc + (imm<<2)	if(rs1 >= rs2) then pc += imm else pc++	B
lw	0b0000011	lw rd, imm(rs1)	rd = mem[rs1+imm], pc++	I
sw	0b0100011	sw rs2, imm(rs1)	mem[addr] = rs2, pc++	S
addi	0b0010011	addi rd, rs1, imm	rd = rs1 + imm, pc++	I
slti	同上	slti rd, rs1, imm	rd = (rs1 < imm) ? 1 : 0, pc++	I
sltiu	同上	sltiu rd, rs1, imm	rd = (rs1 < imm) ? 1 : 0, pc++	I
xori	同上	xori rd, rs1, imm	rd = rs1 ^ imm, pc++	I
ori	同上	ori rd, rs1, imm	rd = rs1	imm, pc++
andi	同上	andi rd, rs1, imm	rd = rs1 & imm, pc++	I
slli	同上	slli rd, rs1, imm	rd = rs1 << imm, pc++	I(5bit)
srl	同上	srl rd, rs1, imm	rd = rs1 >> imm, pc++	I(5bit)
srai	同上	srai rd, rs1, imm	rd = rs1 >>> imm, pc++	I(5bit)
add	0b0110011	add rd, rs1, rs2	rd = rs1 + rs2, pc++	R
sub	同上	sub rd, rs1, rs2	rd = rs1 - rs2, pc++	R
sll	同上	sll rd, rs1, rs2	rd = rs1 << rs2, pc++	R
slt	同上	slt rd, rs1, rs2	rd = (rs1 < rs2) ? 1 : 0, pc++	R
sltu	同上	sltu rd, rs1, rs2	rd = (rs1 < rs2) ? 1 : 0, pc++	R
xor	同上	xor rd, rs1, rs2	rd = rs1 ^ rs2, pc++	R
srl	同上	srl rd, rs1, rs2	rd = rs1 >> rs2, pc++	R
sra	同上	sra rd, rs1, rs2	rd = rs1 >>> rs2, pc++	R
or	同上	or rd, rs1, rs2	rd = rs1	rs2, pc++
and	同上	and rd, rs1, rs2	rd = rs1 & rs2, pc++	R

浮動小数点命令(RV32F改)

命令	opcode	形式	解釈疑似コード	命令フォーマット	レジスタ規定	備考
flw	0b0000111	flw rd, imm(rs1)	rd = mem[rs1+imm]	I	rd:fn,rs1:xn	
fsw	0b0100111	fsw rs2, imm(rs1)	mem[rs1+imm] = rs2	S	rs2:fn,rs1:xn	
fadd	0b1010011	fadd rd, rs1, rs2	rd = rs1 +. rs2	F	rd,rs1,rs2:fn	IPコア実装
fsub	同上	fsub rd, rs1, rs2	rd = rs1 -. rs2	F	rd,rs1,rs2:fn	IPコア実装
fmul	同上	fmul rd, rs1, rs2	rd = rs1 *. rs2	F	rd,rs1,rs2:fn	IPコア実装
fdiv	同上	fdiv rd, rs1, rs2	rd = rs1 /. rs2	F	rd,rs1,rs2:fn	IPコア実装
fsqrt	同上	fsqrt rd, rs1	rd = sqrtf(rs)	F	rd,rs:fn	IPコア実装
fabs	同上	fabs rd, rs1	rd = fabsf(rs)	F	rd,rs:fn	verilog実装
fneg	同上	fneg rd, rs1	rd = -rs	F	rd,rs:fn	verilog実装
feq	同上	feq rd, rs1, rs2	rd = rs1==rs2	F	rd:xn,rs1,rs2:fn	verilog実装
flt	同上	flt rd, rs1, rs2	rd = rs1<rs2	F	rd:xn,rs1,rs2:fn	IPコア実装
fle	同上	fle rd, rs1, rs2	rd = rs1<=rs2	F	rd:xn,rs1,rs2:fn	IPコア実装
itof	同上	itof rd, rs1	rd = (float)rs1	F	rd:fn,rs1:xn	IPコア実装
ftoi	同上	ftoi rd, rs1	rd = roundf(rs2)	F	rd:xn,rs1:fn	IPコア実装
floor	同上	floor rd, rs1	rd = (int)floorf(rs1)	F	rd:xn,rs1:fn	verilog実装
xtof	同上	xtof rd, rs1	rd = rs1(bitコピー)	F	rd:fn,rs1:xn	
ftox	同上	ftox rd, rs1	rd = rs1(bitコピー)	F	rd:xn,rs1:fn	

IO拡張命令

ob

- ・ オペコード: 0b0101011
- ・ funct3: 0b000 (sbと同じ)
- ・ 命令形式: ob rs2

output byteの略。例えば ob x1 とするとx1レジスタの下位8bitを出力

ib

- ・ オペコード: 0b0001011
- ・ funct3: 0b100 (lbuと同じ)
- ・ 命令形式: ib rd

input byteの略。例えば ib x1 とすると8bitの入力を上位24bitゼロ拡張してx1に入れる。

マイクロアーキテクチャについて (担当: 五反田)

- ・ ハーバードアーキテクチャ(メモリ空間等はISAについて参照)
 - 命令メモリはstand aloneのBRAMを使用
 - データメモリはAXI4-Liteのコントローラを経由してBRAMを使用
- ・ 動作周波数: 基本180MHz
 - 最大260MHzまで動作(diff0)を確認。
- ・ 4ステージ構成
 - Fetch,Decode,Execute,Write backの4ステージ
 - 基本各1クロックの合計4クロック構成
 - メモリアクセスはEステージ2クロック
 - 不動小数点演算(IPコア使用およびfloor)はEステージ2~8クロック
 - IO拡張命令はEステージでブロッキング
- ・ IOはUART Lite IPコアを使用
 - Baud Rate: 115200(適宜変更可)
 - パリティ: なし(適宜変更可)
 - IOのエラー処理および投機的実行を行うIPコアのコントローラモジュールを実装
 - IPコアのコントローラはAXI4-StreamプロトコルでCPU本体と通信
- ・ 不動小数点演算には浮動小数点演算用コントローラを使用
- ・ 実行開始用ボタンを作成。
 - チャタリング除去モジュールを実装。

各リソースの使用状況

資源名	使用数	使用率(%)
LUT	3592	1.48
Reg	3232	0.67
BRAM	270.5	45.08

CPU実装の概図

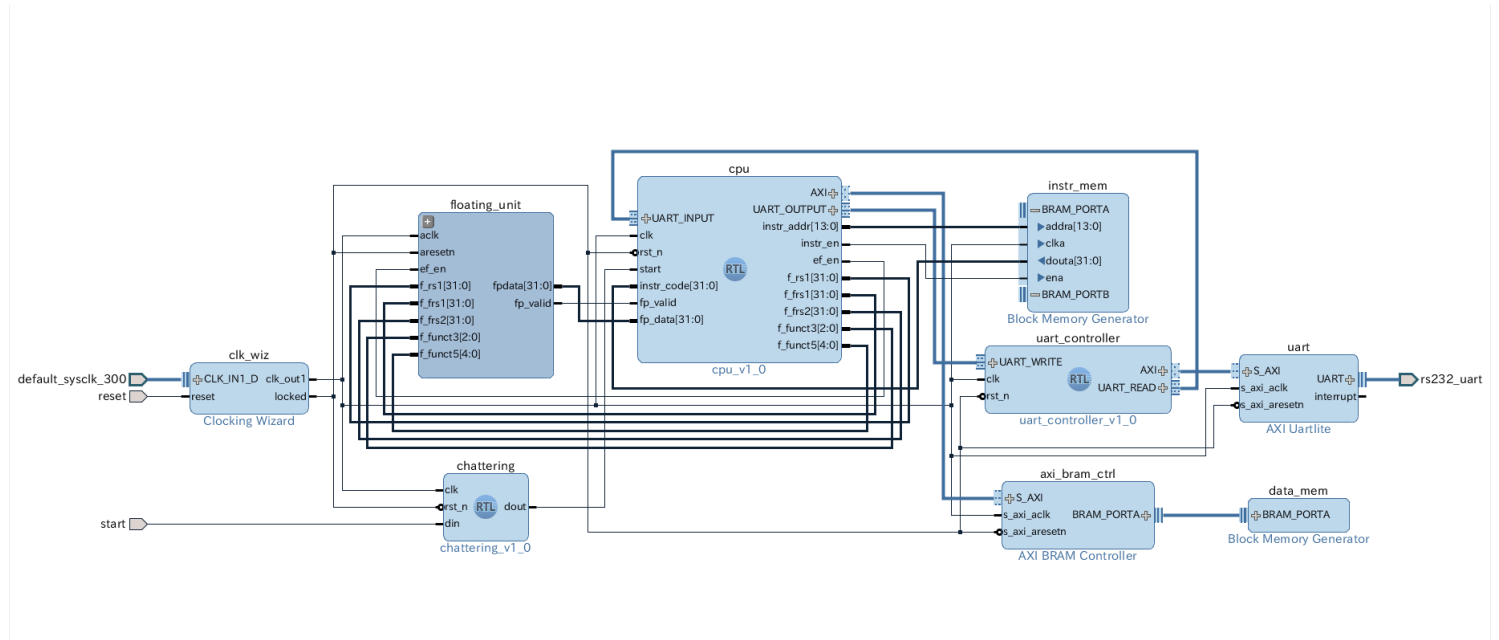


Figure 1: block_design.png

シミュレータについて (担当: 毛利)

概要

- ・ 1 分半ほどで min-rt の実行が終了した。

コマンドラインオプション

- ・ -s ソースファイル名 アセンブリファイルを指定
- ・ -o 出力ファイル名 出力ファイル名 指定しないとsim.outに出力
- ・ -i 入力ファイル名 入力ファイル名 sldを指定する。
- ・ -l ログファイル名 ログファイル名 指定しないとstderrに出力 権限次第で書き込めないかも？らしい

コマンド

- ・ r / run
 - プログラムの全実行
- ・ p / print X (未完成)
 - X に指定したものの値を表示する
 - X ::= pc/pcx/pcd | x0-x31 | f0-f31 | all (すべて表示) | メモリ (int型/float型は別)
 - pc/pcxは16進数で表示。10進数で表示したいならpcd。
- ・ l / log n0 n1
 - 現在の命令から数えてn0番目からn1番目までの命令とその時のレジスタの中身を“simulator.log”に書き出しながら全実行
- ・ o / opcode_next n (未完成)
 - 指定した次のニーモニックまで実行

- ・ n / next n
 - 命令をn個実行
- ・ c / continue n (未完成)
 - 最初から数えてn番目の命令まで実行
- ・ h / help
 - この文章を表示する
- ・ i or initialize
 - 初期化
- ・ q or quit
 - シミュレータの終了