PROJET P-ARM

530 Kess

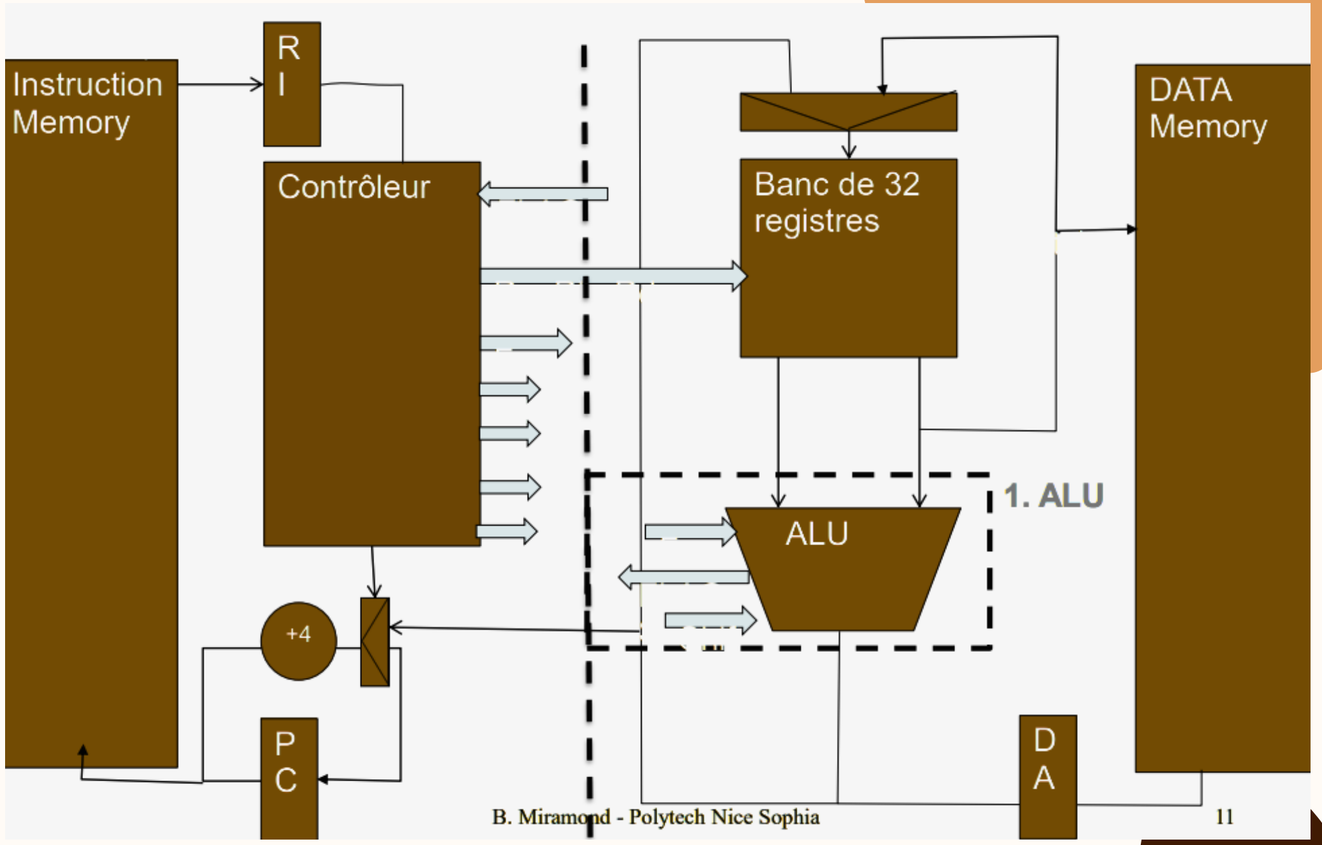
Polytech Nice Sophia - 2024

Antoine Maïstre Rice / Jessica Kahungu / Théo Vidal / Roxane Bacon

SOMMAIRE

PRESENTATION DES COMPOSANTS

- **ALU : En charge des opérations calculatoire**
- **Contrôleur : gere et organise le déroulement d'un fichier**
- **Banc de registres : stocke les registres**



503 Kess - 2024

TABLEAU DE SYNTHESE DES COMPOSANTS

Description	UAL code				Bits																Flags			
	Instruction	opérandes			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	C	V	N	Z
Shift, add, sub, move							opcode																	
Logical Shift Left	LSLS	<Rd>	<Rm>	#imm5	0	0	0	0	0	imm5					Rm			Rd			x		x	x
Logical Shift Right	LSRS	<Rd>	<Rm>	#imm5	0	0	0	0	1	imm5					Rm			Rd			x		x	x
Arithmetic Shift Right	ASRS	<Rd>	<Rm>	#imm5	0	0	0	1	0	imm5					Rm			Rd			x		x	x
Add register	ADDS	<Rd>	<Rn>	<Rm>	0	0	0	1	1	0	0	Rm			Rn			Rd			x	x	x	x
Substract register	SUBS	<Rd>	<Rn>	<Rm>	0	0	0	1	1	0	1	Rm			Rn			Rd			x	x	x	x
Add 3-bit immediate	ADDS	<Rd>	<Rn>	#imm3	0	0	0	1	1	1	0	imm3			Rn			Rd			x	x	x	x
Substract 3-bit immediate	SUBS	<Rd>	<Rn>	#imm3	0	0	0	1	1	1	1	imm3			Rn			Rd			x	x	x	x
Move	MOVS	<Rd>	#imm8		0	0	1	0	0	Rd		imm8											x	x
Compare	CMP	<Rd>	#imm8		0	0	1	0	1	Rd		imm8									x	x	x	x
Add 8-bit immediate	ADDS	<Rdn>	#imm8		0	0	1	1	0	Rd		imm8									x	x	x	x
Substract 8-bit immediate	SUBS	<Rdn>	#imm8		0	0	1	1	1	Rd		imm8									x	x	x	x
Data processing					0	1	0	0	0	0	opcode													
Bitwise AND	ANDS	<Rdn>	<Rm>		0	1	0	0	0	0	0	0	0	0	Rm			Rdn			0		x	x
Exclusive OR	EORS	<Rdn>	<Rm>		0	1	0	0	0	0	0	0	0	1	Rm			Rdn			0	0	x	x
Logical Shift Left	LSLS	<Rdn>	<Rm>		0	1	0	0	0	0	0	0	1	0	Rm			Rdn			x		x	x
Logical Shift Right	LSRS	<Rdn>	<Rm>		0	1	0	0	0	0	0	0	1	1	Rm			Rdn			x		x	x
Arithmetic Shift Right	ASRS	<Rdn>	<Rm>		0	1	0	0	0	0	0	1	0	0	Rm			Rdn					x	
Add with carry	ADDS	<Rdn>	<Rm>		0	1	0	0	0	0	0	1	0	1	Rm			Rdn			x	x	x	x
Substract with carry	SBCS	<Rdn>	<Rm>		0	1	0	0	0	0	0	1	1	0	Rm			Rdn			x	x	x	x
Rotate right	RORS	<Rdn>	<Rm>		0	1	0	0	0	0	0	1	1	1	Rm			Rdn			x		x	x
Set flags on bitwise and	TST	<Rn>	<Rm>		0	1	0	0	0	0	1	0	0	0	<Rm>			Rn			0		x	x
Reverse substract from 0	RSBS	<Rd>	<Rn>	#0	0	1	0	0	0	0	1	0	0	1	<Rn>			Rd			x	x	x	x
Compare registers	CMP	<Rn>	<Rm>		0	1	0	0	0	0	1	0	1	0	<Rm>			Rn			x	x	x	x
Compare negative	CMN	<Rn>	<Rm>		0	1	0	0	0	0	1	0	1	1	<Rm>			Rn			x	x	x	x
Logical OR	ORRS	<Rdn>	<Rm>		0	1	0	0	0	0	1	1	0	0	<Rm>			Rdn			0		x	x
Multiply two registers	MULS	<Rn>	<Rdm>	<Rdm>	0	1	0	0	0	0	1	1	0	1	<Rdm>			Rn					x	x
Bit clear	BICS	<Rdn>	<Rm>		0	1	0	0	0	0	1	1	1	0	<Rm>			Rdn			0		x	x
Bitwise NOT	MVNS	<Rd>	<Rm>		0	1	0	0	0	0	1	1	1	1	<Rm>			Rd			0		x	x
Load/store					1	0	0	1	opcode															
Store Register	STR	<Rt>	#imm8		1	0	0	1	0	Rt		imm8												
Load Register	LDR	<Rt>	#imm8		1	0	0	1	1	Rt		imm8												
Miscellaneous 16-bit instructions					1	0	1	1	opcode															
Add Immediate to SP	ADDSP	#imm7			1	0	1	1	0	0	0	0	0	imm7										
Subtract Immediate from SP	SUBSP	#imm7			1	0	1	1	0	0	0	0	1	imm7										
Branch																								
Equal	BEQ	<#str>			1	1	0	1	0	0	0	0	imm8								Z == 1			
Not equal	BNE	<#str>			1	1	0	1	0	0	0	1	imm8								Z == 0			
Carry	BCS/BHS	<#str>			1	1	0	1	0	0	1	0	imm8								C == 1			
No carry	BCC/BLO	<#str>			1	1	0	1	0	0	1	1	imm8								C == 0			
Negative	BMI	<#str>			1	1	0	1	0	1	0	0	imm8								N == 1			
Positive or zero	BPL	<#str>			1	1	0	1	0	1	0	1	imm8								N == 0			
Overflow	BVS	<#str>			1	1	0	1	0	1	1	0	imm8								V == 1			
No overflow	BVC	<#str>			1	1	0	1	0	1	1	1	imm8								V == 0			
Greater than (unsigned)	BHI	<#str>			1	1	0	1	1	0	0	0	imm8								C == 1 et Z == 0			
Less Than or Equal (Unsigned)	BLS	<#str>			1	1	0	1	1	0	0	1	imm8								C == 0 ou Z == 1			
Greater Than or Equal (Signed)	BGE	<#str>			1	1	0	1	1	0	1	0	imm8								N == V			
Less Than (Signed)	BLT	<#str>			1	1	0	1	1	0	1	1	imm8								N != V			
Greater Than (Signed)	BGT	<#str>			1	1	0	1	1	1	0	0	imm8								Z == 0 et N == V			
Less Than or Equal (Signed)	BLE	<#str>			1	1	0	1	1	1	0	1	imm8								Z == 1 ou N != V			
Always true	BAL	<#str>			1	1	0	1	1	1	1	0	imm8											
B : Unconditional Branch	B	#imm11			1	1	1	0	0	imm11														

TESTS UNITAIRES

Composants	Tests passés	Tests non passés
ALU	1	0
Conditional	1	0
Data_Processing	1	0
Opcode_Decoder	1	0
SASM	1	0
SP_address	1	0
Taux de couverture	100%	0%

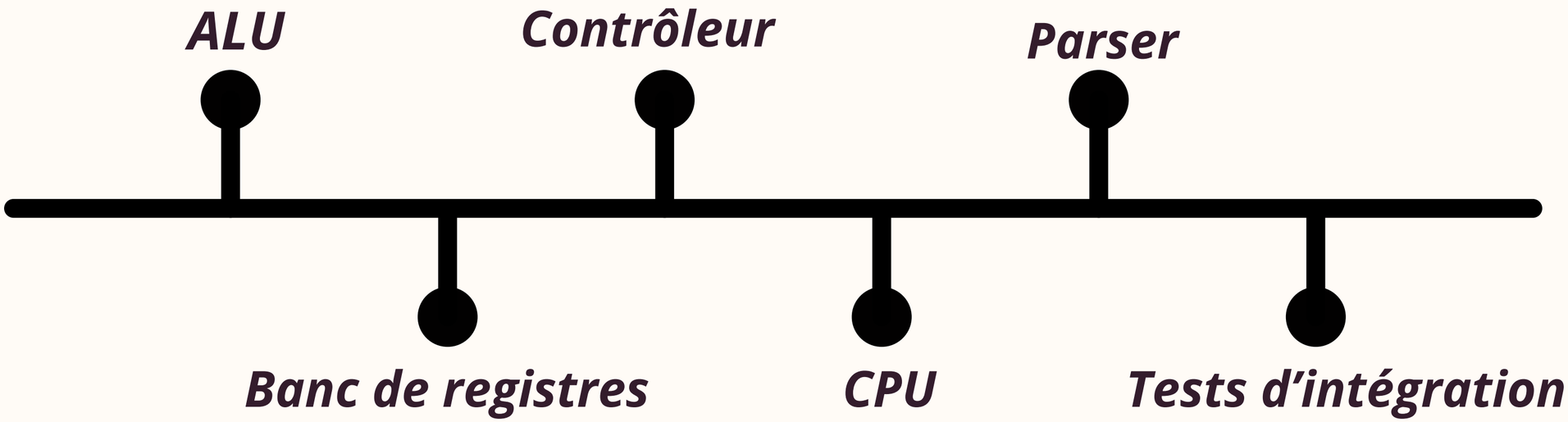
TESTS D'INTÉGRATION

Codes ASM	Test passé /1	Non testé /1
Conditional	1	0
DP_1_4	1	0
DP_5_10	1	0
DP_11_12	1	0
DP_13_16	1	0
Load_store	1	0
SP	1	0
SASM_1_4	1	0
SASM_5_8	1	0
Taux de couverture	**100%**	**0%**

TESTS D'ASSEMBLEUR

Codes C	Test asm passé /1	Tests logisim passé /1	Non testé /1
Calckeyb	1	1	0
Calculator	1	1	0
Simple_add	1	1	0
Testfp	1	0	0,5
Tty	1	1	0
My own test	1	1	0
Taux de couverture	100%	84%	8%

ORGANISATION



Des tests unitaires ont été réalisés sur de nombreux composants au fur et à mesure

COMPILATEUR ASM -> BIN

Conception et démonstration

Langage utilisé: Typescript

```
async function main() {
  const filename = process.argv[2];
  if (!filename) return console.error("No input file specified.");
  const fileStream = fs.createReadStream(filename);
  const rl = readline.createInterface({ input: fileStream, crlfDelay

  const lines: string[] = [];
  for await (const line of rl) {
    lines.push(line);
  }

  const labels = findLabels(lines);
  const hexInstructions = buildHexInstruction(lines, labels);
  const outputFilename = path.basename(filename, path.extname(filename)) + '.bin';
  const outputStream = fs.createWriteStream(outputFilename);
  outputStream.write("v2.0 raw\n");

  for (const hexInstruction of hexInstructions) {
    outputStream.write(`${hexInstruction} `);
  }

  outputStream.end();
  console.log(`Assembled file written to ${outputFilename}`);
}
main().catch(err => console.error(err));
```

```
export function asmArithmetic3(opcode: string, rd: string, imm: string): string
  const immEncoded = 8;
  const opcodeBinary = {
    'MOVS': '00',
    'CMP': '01',
    'ADDS': '10',
    'SUBS': '11',
  }[opcode];
  return `001 ${opcodeBinary} ${registerToBinary(rd)} ${immToBinary(imm, immEnco
}
```

COMPILATEUR ASM -> BIN

Tests unitaires (avec Jest)

```
describe('Data processing Assembly Line to Hex Conversion', () => {...
describe('Load Store Assembly Line to Hex Conversion', () => {...
describe('Miscellaneous Assembly Line to Hex Conversion', () => {...
describe('Shift add sub mov Assembly Line to Hex Conversion', () =>
describe('Branch Assembly Line to Hex Conversion', () => {...
```

```
describe('Load Store Assembly Line to Hex Conversion', () =>
{ test.each([
    ['movs r0, #170', '20AA'],
    ['movs r1, #255', '21FF'],
    ['add sp, #16', 'B004'],
    ['str r0, [sp, #4]', '9001'],
    ['str r1, [sp, #0]', '9100'],
    ['sub sp, #4', 'B081'],
    ['ldr r2, [sp, #4]', '9A01'],

  ])('converts "%s" to hex "%s"', (input, expected) => {
    const binaryOutput = asm(input, new Map(), 0);
    const hexOutput = binaryToHex(binaryOutput);
    expect(hexOutput).toBe(expected);
  });
});
```

```
> jest --silent

 PASS  dist/__tests__/branch.test.js
 PASS  __tests__/branch.test.ts
 PASS  __tests__/asm.test.ts
 PASS  dist/__tests__/asm.test.js

Test Suites: 4 passed, 4 total
Tests:       172 passed, 172 total
Snapshots:   0 total
Time:        0.89 s, estimated 1 s
```

PROBLÈMES RENCONTRÉS

Problème 1

Compréhension du sujet (prise en main de Logisim, compréhension des documents, etc...)

Problème 2

Incompréhension au niveau de certains branchements Logisim

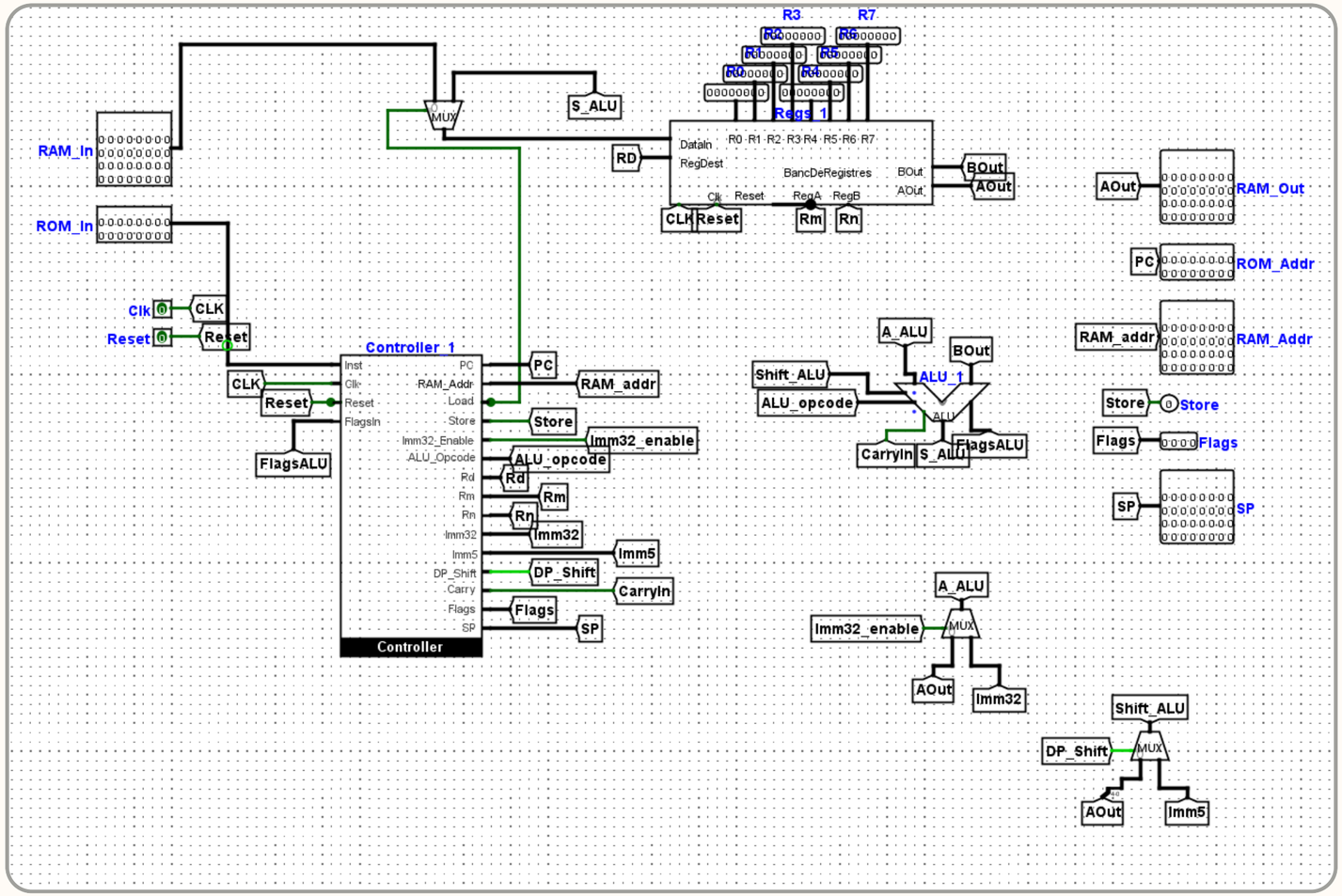
Problème 3

Conditional & Shift Add Sub
Mov compliqué car Excel mal organisé donc perte de temps considérable

Problème 4

Difficulté à comprendre les conditions et labels
Des instructions inconnues / manque de solutions (Versions de CLANG...)

DEMONSTRATION



CONCLUSION

Antoine Maïstre-Rice
Jessica Kahungu
Théo Vidal
Roxane Bacon