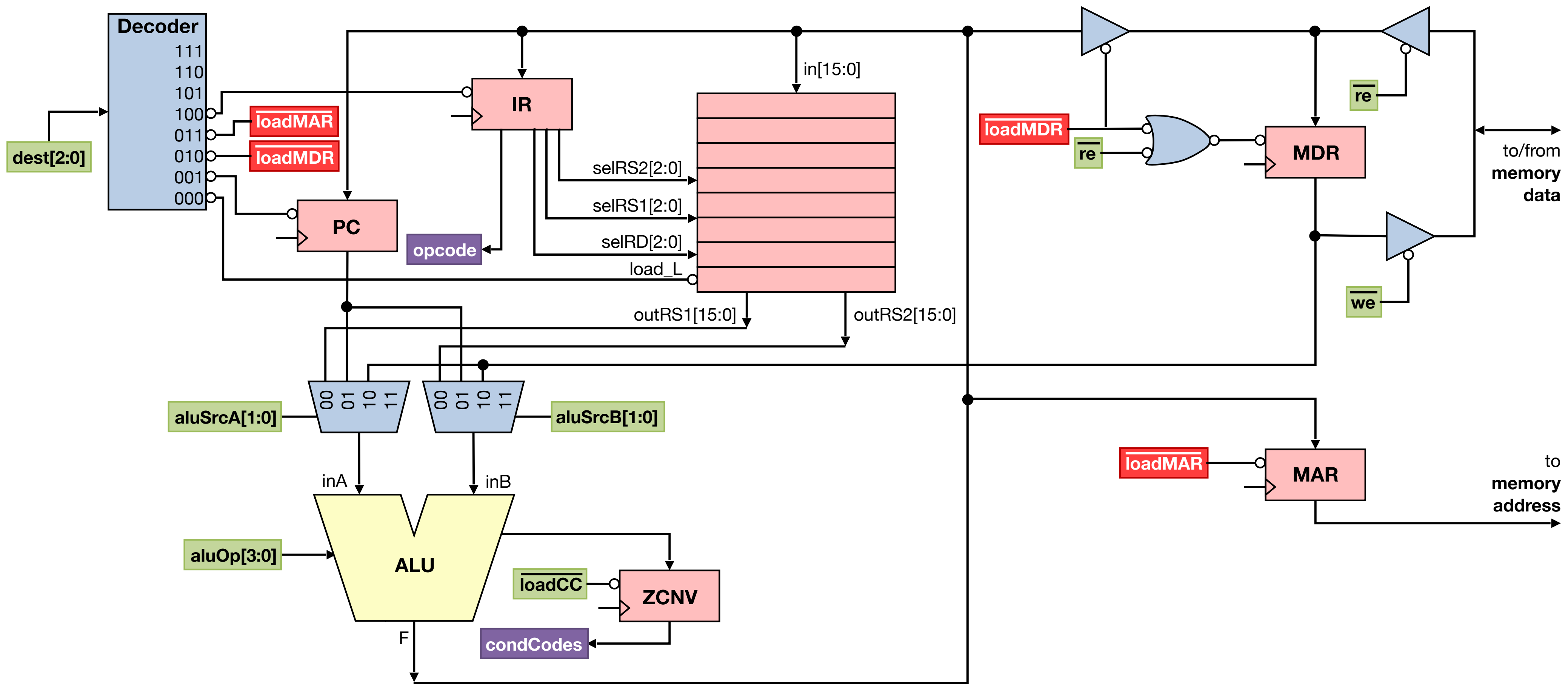
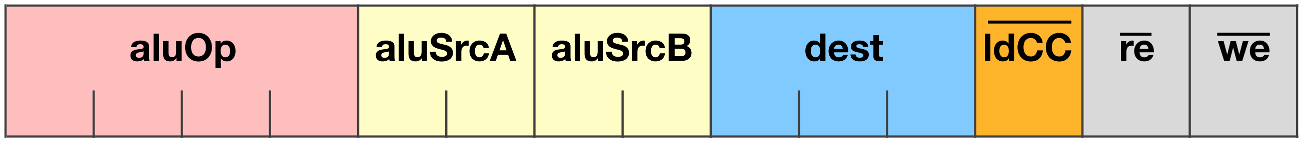

Decoder
111
110
101
100
011
010
001
000
dest[2:0]
$\overline{\text{loadMAR}}$
$\overline{\text{loadMDR}}$
IR
PC
opcode
in[15:0]
selRS2[2:0]
selRS1[2:0]
selRD[2:0]
load_L
outRS1[15:0]
outRS2[15:0]
00 01 10 11
00 01 10 11
aluSrcA[1:0]
aluSrcB[1:0]
inA
inB
aluOp[3:0]
ALU
F
$\overline{\text{loadCC}}$
ZCNV
condCodes
$\overline{\text{loadMDR}}$
$\overline{\text{re}}$
$\overline{\text{re}}$
MDR
$\overline{\text{we}}$
to/from memory data
$\overline{\text{loadMAR}}$
MAR
to memory address

Microinstruction Definitions



```
typedef enum logic[3:0]{
  F_A_PLUS_B  = 4'b0000,
  F_A_MINUS_B = 4'b0001,
  F_A         = 4'b0010,
  F_B         = 4'b0011,
  F_A_PLUS_2  = 4'b0100,
  F_A_LT_B    = 4'b0101,
  // 4'b0110 reserved
  // 4'b0111 reserved
  F_A_NOT     = 4'b1000,
  F_A_AND_B   = 4'b1001,
  F_A_OR_B    = 4'b1010,
  F_A_XOR_B   = 4'b1011,
  F_A_SHL     = 4'b1100,
  // 4'b1101 reserved
  F_A_LSHR    = 4'b1110,
  F_A_ASHR    = 4'b1111,
  F_UNDEF     = 4'bxxxx
} alu_op_t;
```

	F		F
0000	A + B	1000	not (A)
0001	A - B	1001	A • B
0010	A	1010	A or B
0011	B	1011	A ⊕ B
0100	A + 2	1100	shl (A)
0101	A < B	1101	(none)
0110	(none)	1110	lshr (A)
0111	(none)	1111	ashr (A)

```
typedef enum logic [1:0]{
  MUX_REG   = 2'b00,
  MUX_PC    = 2'b01,
  MUX_MDR   = 2'b10,
  MUX_UNDEF = 2'bxx
} alu_mux_t;
```

	op
00	RegFile
01	PC
10	MDR

```
typedef enum logic [2:0]{
  DEST_REG    = 3'b000,
  DEST_PC     = 3'b001,
  DEST_MDR    = 3'b010,
  DEST_MAR    = 3'b011,
  DEST_IR     = 3'b100,
  DEST_NONE   = 3'b111,
  DEST_UNDEF  = 3'bxxx
} dest_sel_t;
```

	dest
000	RegFile
001	PC
010	MDR
011	MAR
100	IR
101	(none)
110	(none)
111	(none)

```
typedef enum logic{
  LOAD_CC = 1'b0,
  NO_LOAD = 1'b1
} cond_code_t;
```

	op
0	Load condition codes
1	No load

```
typedef enum logic{
  MEM_RD = 1'b0,
  NO_RD  = 1'b1
} rd_enable_t;
```

	op
0	Write: Mem[MAR] ← MDR
1	No write

```
typedef enum logic{
  MEM_WR = 1'b0,
  NO_WR  = 1'b1
} wr_enable_t;
```

	op
0	Read: MDR ← Mem[MAR]
1	No read

5. RISC240 Assembly Instructions

ADD rd, rs1, rs2

Addition

Semantics: `rd ← rs1 + rs2`
CC Flags: `ZCNV - set normally`

Encoding:

15 9	8 6	5 3	2 0
0000 000	rd	rs1	rs2

Cycles: 5

ADDI rd, rs1, imm

Add Immediate

Semantics: `rd ← rs1 + imm`
CC Flags: `ZCNV - set normally`

Encoding:

15 9	8 6	5 3	2 0
0011 000	rd	rs1	000
imm			

Cycles: 7

AND rd, rs1, rs2

Bitwise AND

Semantics: `rd ← rs1 · rs2`
CC Flags: `ZN - set normally`
`C = 0`
`V = 0`

Encoding:

15 9	8 6	5 3	2 0
1001 000	rd	rs1	rs2

Cycles: 5

BRA addr

Branch Always

Semantics: `PC ← addr`
CC Flags: `not changed`

Encoding:

15 9	8 6	5 3	2 0
1111 100	000	000	000
addr			

Cycles: 7

BRC addr

Branch If Carry

Semantics: `if(C) PC ← addr`
CC Flags: `not changed`

Encoding:

15 9	8 6	5 3	2 0
1010 100	000	000	000
addr			

Cycles: 7 if taken; 6 if not taken

BRN addr

Branch If Negative

Semantics: `if(N) PC ← addr`
CC Flags: `not changed`

Encoding:

15 9	8 6	5 3	2 0
1001 100	000	000	000
addr			

Cycles: 7 if taken; 6 if not taken

BRNZ addr
Branch If Negative or Zero

Semantics: `if(N OR Z) PC ← addr`
CC Flags: `not changed`

Encoding:

15 9	8 6	5 3	2 0
1101 100	000	000	000
addr			

Cycles: 7 if taken; 6 if not taken

BRV addr
Branch If Overflow

Semantics: `if(V) PC ← addr`
CC Flags: `not changed`

Encoding:

15 9	8 6	5 3	2 0
1011 100	000	000	000
addr			

Cycles: 7 if taken; 6 if not taken

BRZ addr
Branch If Zero

Semantics: `if(Z) PC ← addr`
CC Flags: `not changed`

Encoding:

15 9	8 6	5 3	2 0
1100 100	000	000	000
addr			

Cycles: 7 if taken; 6 if not taken

LI rd, imm
Load Immediate

Semantics: `rd ← imm`
CC Flags: `ZCNV - set normally`

Encoding:

15 9	8 6	5 3	2 0
0011 000	rd	000	000
imm			

Cycles: 7

Notes: implemented in hardware as ADDI rd, r0, imm

LW rd, rs1, imm
Load Word

Semantics: `rd ← M[rs1 + imm]`
CC Flags: `ZN - set normally`
`C = 0`
`V = 0`

Encoding:

15 9	8 6	5 3	2 0
0010 100	rd	rs1	000
imm			

Cycles: 9

MV rd, rs1
Move Register

Semantics: `rd ← rs1`
CC Flags: `not changed`

Encoding:

15 9	8 6	5 3	2 0
0010 000	rd	rs1	000

Cycles: 5

NOT rd, rs1
Bitwise NOT

Encoding:

15 9	8 6	5 3	2 0
1000 000	rd	rs1	000

Semantics: `rd ← NOT rs1`

CC Flags:
```
ZN - set normally
C = 0
V = 0
```

Cycles: 5

OR rd, rs1, rs2
Bitwise OR

Encoding:

15 9	8 6	5 3	2 0
1010 000	rd	rs1	rs2

Semantics: `rd ← rs1 OR rs2`

CC Flags:
```
ZN - set normally
C = 0
V = 0
```

Cycles: 5

SLL rd, rs1, rs2
Shift Left Logical

Encoding:

15 9	8 6	5 3	2 0
1100 000	rd	rs1	rs2

Semantics: `rd ← rs1 << rs2[3:0]`

CC Flags:
```
ZN - set normally
C = 0
V = 0
```

Cycles: 5

Notes: for a shift amount of *n*, shifts in *n* zeros from the right

---

SLLI rd, rs1, shamt
Shift Left Logical Immediate

Encoding:

15 9	8 6	5 3	2 0
1100 001	rd	rs1	000
shamt			

Semantics: `rd ← rs1 << shamt[3:0]`

CC Flags:
```
ZN - set normally
C = 0
V = 0
```

Cycles: 7

Notes: for a shift amount of *n*, shifts in *n* zeros from the right

---

SLT rd, rs1, rs2
Set If Less Than

Semantics: `if(rs1 < rs2) rd ← 16’b1`
`else rd ← 16’b0`
CC Flags: `ZCNV - set based on result of`
`(rs1 - rs2)`

Notes: two’s complement comparison

Encoding:

15 9	8 6	5 3	2 0
0101 000	rd	rs1	rs2

Cycles: 6

SLTI rd, rs1, imm
Set If Less Than Immediate

Semantics: `if(rs1 < imm) rd ← 16’b1`
`else rd ← 16’b0`
CC Flags: `ZCNV - set based on result of`
`(rs1 - imm)`

Notes: two’s complement comparison

Encoding:

15 9	8 6	5 3	2 0
0101 001	rd	rs1	000
imm			

Cycles: 8

SRA rd, rs1, rs2
Shift Right Arithmetic

Semantics: `rd ← rs1 >>> rs2[3:0]`
CC Flags: `ZN - set normally`
`C = 0`
`V = 0`

Notes: for a shift amount of *n*, shifts in *n* copies of `rs1[15]` from the left

Encoding:

15 9	8 6	5 3	2 0
1111 000	rd	rs1	rs2

Cycles: 5

SRAI rd, rs1, shamt
Shift Right Arithmetic Immediate

Semantics: `rd ← rs1 >>> shamt[3:0]`
CC Flags: `ZN - set normally`
`C = 0`
`V = 0`

Notes: for a shift amount of *n*, shifts in *n* copies of `rs1[15]` from the left

Encoding:

15 9	8 6	5 3	2 0
1111 001	rd	rs1	000
shamt			

Cycles: 7

SRL rd, rs1, rs2
Shift Right Logical

Encoding:

15 9	8 6	5 3	2 0
`1110 000`	`rd`	`rs1`	`rs2`

Semantics: `rd ← rs1 >> rs2[3:0]`
CC Flags: `ZN - set normally`
`C = 0`
`V = 0`

Cycles: 5

Notes: for a shift amount of *n*, shifts in *n* zeros from the left

---

SRLI rd, rs1, shamt
Shift Right Logical Immediate

Encoding:

15 9	8 6	5 3	2 0
`1110 001`	`rd`	`rs1`	`000`
`shamt`			

Semantics: `rd ← rs1 >> shamt[3:0]`
CC Flags: `ZN - set normally`
`C = 0`
`V = 0`

Cycles: 7

Notes: for a shift amount of *n*, shifts in *n* zeros from the left

---

STOP
Stop Execution

Encoding:

15 9	8 6	5 3	2 0
`1111 111`	`000`	`000`	`000`

Semantics: `PC ← PC`
CC Flags: `not changed`

Cycles: 5

Notes: used to halt the CPU when program execution is finished, by looping infinitely on the STOP instruction until a reset signal is asserted

---

SUB rd, rs1, rs2
Subtraction

Encoding:

15 9	8 6	5 3	2 0
`0001 000`	`rd`	`rs1`	`rs2`

Semantics: `rd ← rs1 - rs2`
CC Flags: `ZCNV - set normally`

Cycles: 5

SW rs1, rs2, imm
Store Word

Encoding:

15 9	8 6	5 3	2 0
`0011 100`	`000`	`rs1`	`rs2`
`imm`			

Semantics: `M[rs1 + imm] ← rs2`
CC Flags: `ZN - set normally`
`C = 0`
`V = 0`

Cycles: 9

XOR rd, rs1, rs2
Bitwise XOR

Semantics: `rd` $\leftarrow$ `rs1` $\oplus$ `rs2`

CC Flags:
```
ZN - set normally
C = 0
V = 0
```

Encoding:

15 … 9	*8 … 6*	*5 … 3*	*2 … 0*
1011 000	rd	rs1	rs2

Cycles: 5