We acknowledge and pay our respects to the Kaurna people, the traditional custodians whose ancestral lands we gather on.

We acknowledge the deep feelings of attachment and relationship of the Kaurna people to country and we respect and value their past, present and ongoing connection to the land and cultural beliefs.

Computer Systems

Lecture 10: Assembler Review and Exercise

THE UNIVERSITY of ADELAIDE

Exercise

When this program has finished assembling, what is the resulting machine code?

```
// Computes 1+...+RAM[0]
// And stored the sum in RAM[1]
    @i
    M=1   // i = 1
    @sum
    M=0   // sum = 0
(LOOP)
    @i    // if i>RAM[0] goto WRITE
    D=M
    @R0
    D=D-M
    @WRITE
    D;JGT
    @i    // sum += i
    D=M
    @sum
    M=D+M
    @i    // i++
    M=M+1
    @LOOP // goto LOOP
    0;JMP
(WRITE)
    @sum
    D=M
    @R1
    M=D  // RAM[1] = the sum
(END)
    @END
    0;JMP
```

THE UNIVERSITY of ADELAIDE

Symbol Table:
Pre-defined Symbols

```
// Computes 1+...+RAM[0]
// And stored the sum in RAM[1]
    @i
    M=1   // i = 1
    @sum
    M=0   // sum = 0
(LOOP)
    @i    // if i>RAM[0] goto WRITE
    D=M
    @R0
    D=D-M
    @WRITE
    D;JGT
    @i    // sum += i
    D=M
    @sum
    M=D+M
    @i    // i++
    M=M+1
    @LOOP // goto LOOP
    0;JMP
(WRITE)
    @sum
    D=M
    @R1
    M=D  // RAM[1] = the sum
(END)
    @END
    0;JMP
```

THE UNIVERSITY of ADELAIDE

Symbol Table: Pre-defined Symbols

Symbol	Address
R0	0
R1	1
…	…
R15	15
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
SCREEN	16384
KBD	24576

```
// Computes 1+...+RAM[0]
// And stored the sum in RAM[1]
    @i
    M=1   // i = 1
    @sum
    M=0   // sum = 0
(LOOP)
    @i    // if i>RAM[0] goto WRITE
    D=M
    @R0
    D=D-M
    @WRITE
    D;JGT
    @i    // sum += i
    D=M
    @sum
    M=D+M
    @i    // i++
    M=M+1
    @LOOP // goto LOOP
    0;JMP
(WRITE)
    @sum
    D=M
    @R1
    M=D  // RAM[1] = the sum
(END)
    @END
    0;JMP
```

THE UNIVERSITY of ADELAIDE

Code Clean-up

```
// Computes 1+...+RAM[0]
// And stored the sum in RAM[1]
    @i
    M=1   // i = 1
    @sum
    M=0   // sum = 0
(LOOP)
    @i    // if i>RAM[0] goto WRITE
    D=M
    @R0
    D=D-M
    @WRITE
    D;JGT
    @i    // sum += i
    D=M
    @sum
    M=D+M
    @i    // i++
    M=M+1
    @LOOP // goto LOOP
    0;JMP
(WRITE)
    @sum
    D=M
    @R1
    M=D  // RAM[1] = the sum
(END)
    @END
    0;JMP
```

THE UNIVERSITY of ADELAIDE

Code Clean-up

```
@i
M=1
@sum
M=0
(LOOP)
@i
D=M
@R0
D=D-M
@WRITE
D;JGT
@i
D=M
@sum
M=D+M
@i
M=M+1
@LOOP
0;JMP
(WRITE)
@sum
D=M
@R1
M=D
(END)
@END
0;JMP
```

THE UNIVERSITY of ADELAIDE

Symbol Table: After the 1st Pass

```
@i
M=1
@sum
M=0
(LOOP)
@i
D=M
@R0
D=D-M
@WRITE
D;JGT
@i
D=M
@sum
M=D+M
@i
M=M+1
@LOOP
0;JMP
(WRITE)
@sum
D=M
@R1
M=D
(END)
@END
0;JMP
```

Symbol Table: After the 1st Pass

Symbol	Address
Predefined symbols…	0…15 …
LOOP	4
WRITE	18
END	22

```
0  @i
1  M=1
2  @sum
3  M=0
↓  (LOOP)
4  @i
5  D=M
6  @R0
7  D=D-M
8  @WRITE
9  D;JGT
10 @i
11 D=M
12 @sum
13 M=D+M
14 @i
15 M=M+1
16 @LOOP
17 0;JMP
↓  (WRITE)
18 @sum
19 D=M
20 @R1
21 M=D
↓  (END)
22 @END
23 0;JMP
```

THE UNIVERSITY of ADELAIDE

Symbol Table: During the 2nd Pass

Symbol	Address
Predefined symbols…	0…15 …
LOOP	4
WRITE	18
END	22

```
0  @i
1  M=1
2  @sum
3  M=0
4  @i
5  D=M
6  @0
7  D=D-M
8  @18
9  D;JGT
10 @i
11 D=M
12 @sum
13 M=D+M
14 @i
15 M=M+1
16 @4
17 0;JMP
18 @sum
19 D=M
20 @1
21 M=D
22 @22
23 0;JMP
```

Symbol Table: During the 2nd Pass

Symbol	Address
Predefined symbols…	0…15 …
LOOP	4
WRITE	18
END	22
i	16
sum	17

```
0  @i
1  M=1
2  @sum
3  M=0
4  @i
5  D=M
6  @0
7  D=D-M
8  @18
9  D;JGT
10 @i
11 D=M
12 @sum
13 M=D+M
14 @i
15 M=M+1
16 @4
17 0;JMP
18 @sum
19 D=M
20 @1
21 M=D
22 @22
23 0;JMP
```

THE UNIVERSITY of ADELAIDE

During Code Generation

Symbol	Address
Predefined symbols…	0…15 …
LOOP	4
WRITE	18
END	22
i	16
sum	17

```
0  @16
1  M=1
2  @17
3  M=0
4  @16
5  D=M
6  @0
7  D=D-M
8  @18
9  D;JGT
10 @16
11 D=M
12 @17
13 M=D+M
14 @16
15 M=M+1
16 @4
17 0;JMP
18 @17
19 D=M
20 @1
21 M=D
22 @22
23 0;JMP
```

Translating / assembling A-instructions

Symbolic: `@`*value* // Where *value* is either a non-negative decimal number
// or a symbol referring to such number.

value (v = 0 or 1)

Binary:

0	v	v	v	v	v	v	v	v	v	v	v	v	v	v	v

<u>Translation to binary:</u>

- If *value* is a non-negative decimal number, simple
- If *value* is a symbol…

Translating / assembling C-instructions

Symbolic: *dest=comp;jump* // Either the *dest* or *jump* fields may be empty.
// If *dest* is empty, the "=" is ommitted;
// If *jump* is empty, the ";" is omitted.

comp *dest* *jump*

Binary: 1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3

(when a=0) *comp*	c1	c2	c3	c4	c5	c6	(when a=1) *comp*
0	1	0	1	0	1	0	
1	1	1	1	1	1	1	
-1	1	1	1	0	1	0	
D	0	0	1	1	0	0	
A	1	1	0	0	0	0	M
!D	0	0	1	1	0	1	
!A	1	1	0	0	0	1	!M
-D	0	0	1	1	1	1	
-A	1	1	0	0	1	1	-M
D+1	0	1	1	1	1	1	
A+1	1	1	0	1	1	1	M+1
D-1	0	0	1	1	1	0	
A-1	1	1	0	0	1	0	M-1
D+A	0	0	0	0	1	0	D+M
D-A	0	1	0	0	1	1	D-M
A-D	0	0	0	1	1	1	M-D
D&A	0	0	0	0	0	0	D&M
D|A	0	1	0	1	0	1	D|M

d1	d2	d3	*Mnemonic*	*Destination (where to store the computed value)*
0	0	0	null	The value is not stored anywhere
0	0	1	M	Memory[A] (memory register addressed by A)
0	1	0	D	D register
0	1	1	MD	Memory[A] and D register
1	0	0	A	A register
1	0	1	AM	A register and Memory[A]
1	1	0	AD	A register and D register
1	1	1	AMD	A register, Memory[A], and D register

j1 (*out* <0)	j2 (*out* $=0$)	j3 (*out* >0)	Mnemonic	Effect
0	0	0	null	No jump
0	0	1	JGT	If *out* >0 jump
0	1	0	JEQ	If *out* $=0$ jump
0	1	1	JGE	If *out* ≥ 0 jump
1	0	0	JLT	If *out* <0 jump
1	0	1	JNE	If *out* $\neq 0$ jump
1	1	0	JLE	If *out* ≤ 0 jump
1	1	1	JMP	Jump

Question 1

1 pts

Look at the following (incomplete) diagram of the Hack CPU. Look at the wire pointed to by the large red arrow.

Where does the signal on this wire come from and what action does this signal trigger?

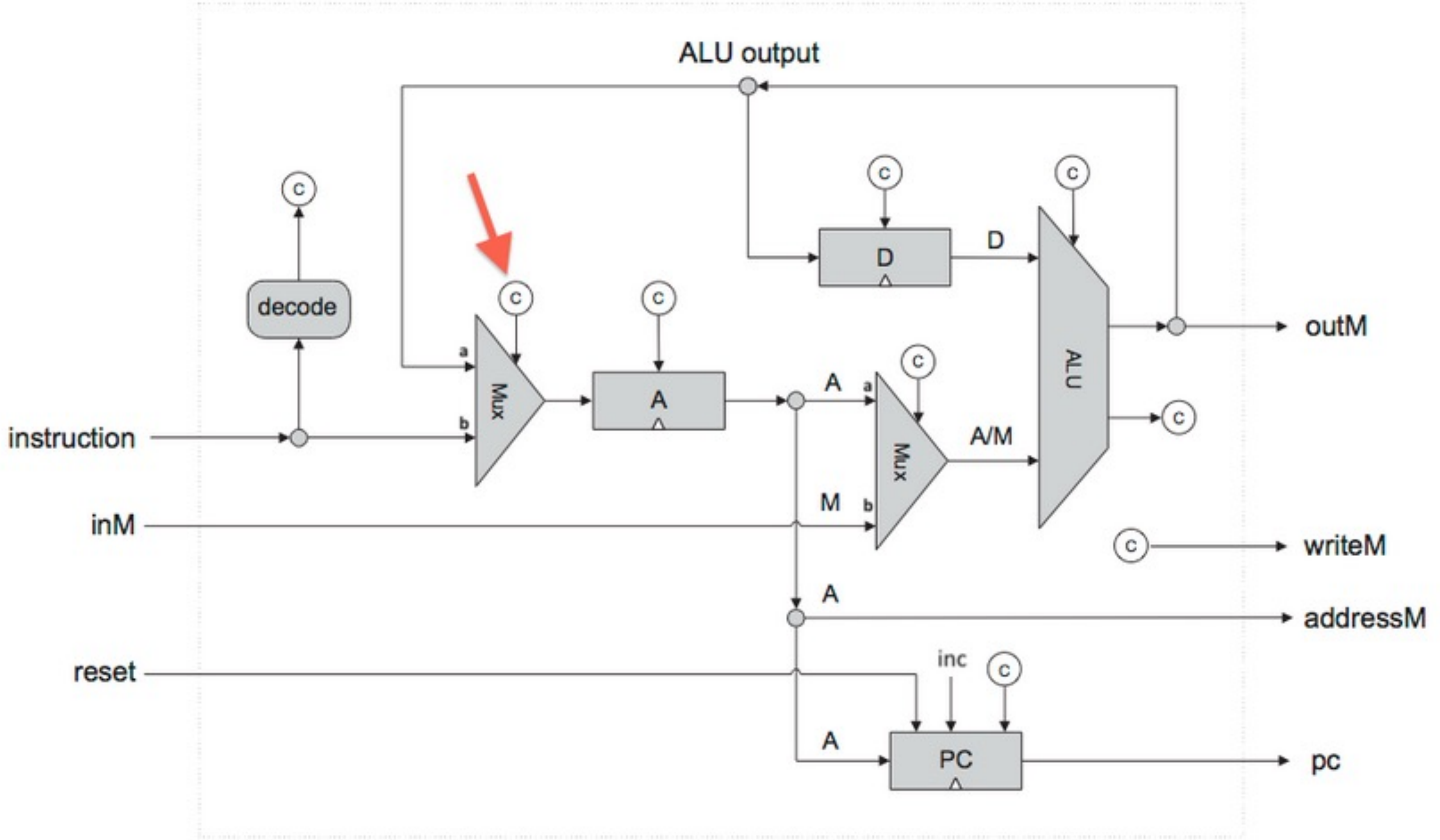

Figure 5.9 Proposed CPU implementation. The diagram shows only *data* and *address paths*, namely, wires that carry data and addresses from one place to another. The diagram does not show the CPU's *control logic*, except for inputs and outputs of control bits, labeled with a circled "c". Thus it should be viewed as an incomplete chip diagram.

THE UNIVERSITY *of* ADELAIDE

Question 2

1 pts

Look at the following (incomplete) diagram of the Hack CPU. Look at the wire (and it is a single wire) pointed to by the large red arrow.

Where does the signal on this wire come from and what action does this signal trigger?

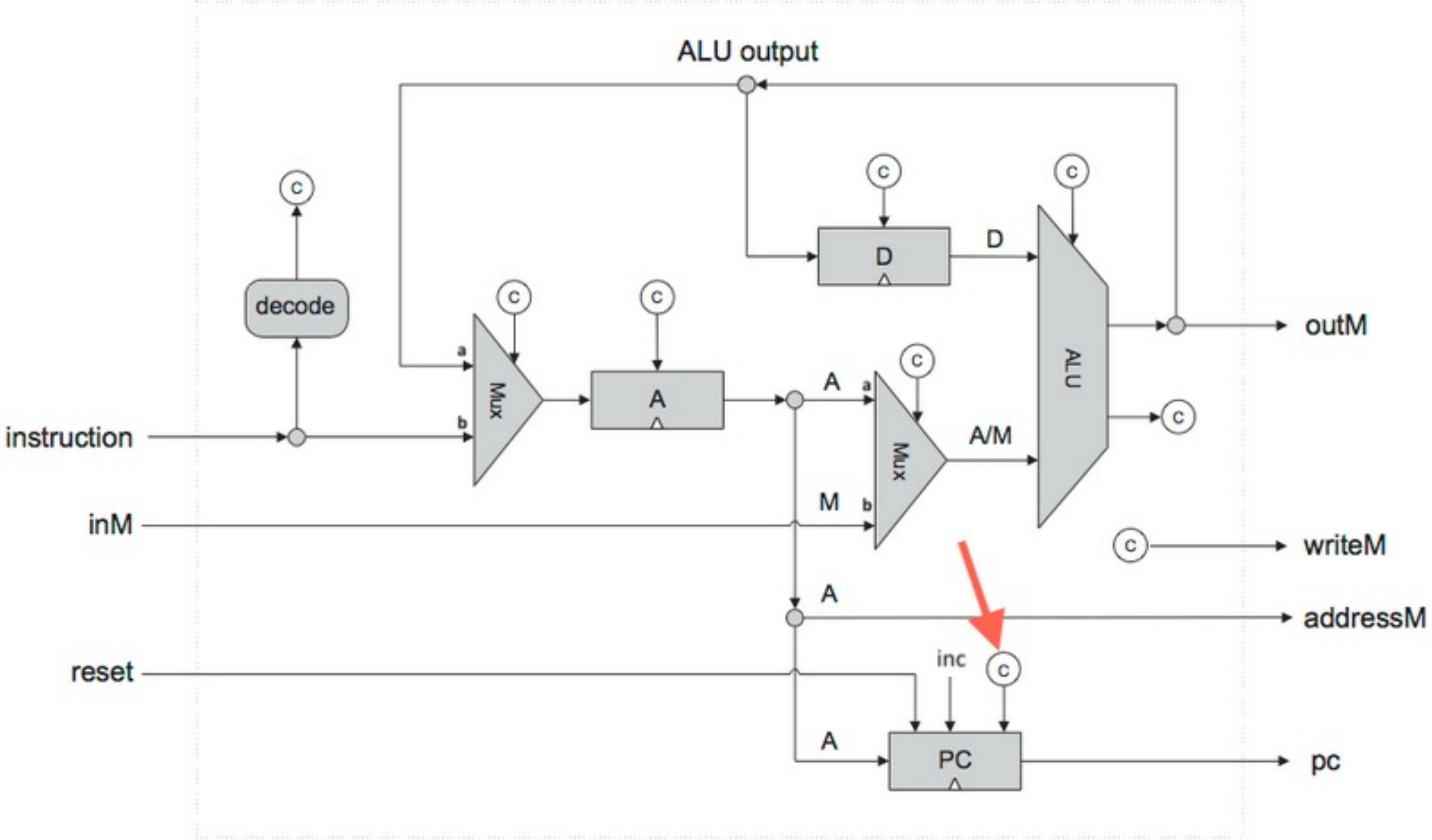

Figure 5.9 Proposed CPU implementation. The diagram shows only *data* and *address paths*, namely, wires that carry data and addresses from one place to another. The diagram does not show the CPU's *control logic*, except for inputs and outputs of control bits, labeled with a circled "c". Thus it should be viewed as an incomplete chip diagram.

THE UNIVERSITY of ADELAIDE

Jump Unit (Assignment 4)

The Jump Unit provides a method for us to tell the CPU when to Jump to a different part of the running program.

This will allow us to have loops and conditional statements. In the Hack architecture, this is achieved using the Jump bits of the C-instruction, and comparing these to the ALU's output using its status bits.

- **The j1, j2 and j3 bits of the C-instruction tell the CPU whether to perform a Jump if a condition is met.**
  - If the j1 bit is set true, a jump should occur if the ALU's output is less than 0
  - If the j2 bit is set true, a jump should occur if the ALU's output is equal to 0
  - If the j3 bit is set true, a jump should occur if the ALU's output is greater than 0
- We can combine these bits:
  - If j1 and j2 are both true, a jump occurs if either the ALU's output is less than 0, or if the ALU's output is equal to 0
  - If all 3 bits are true, a jump always occurs (because the ALU's output is either < 0 or > 0 or 0)
  - If none of 3 bits are true, a jump never occurs
- **We can determine whether the ALU's output is < 0 or > 0 or 0 by checking the ALU's status bits:**
  - the zr bit will be true if the ALU's output is 0.
  - the ng bit will be true if the ALU's output is < 0.

Question 2

1 pts

Look at the following (incomplete) diagram of the Hack CPU. Look at the wire (and it is a single wire) pointed to by the large red arrow.

Where does the signal on this wire come from and what action does this signal trigger?

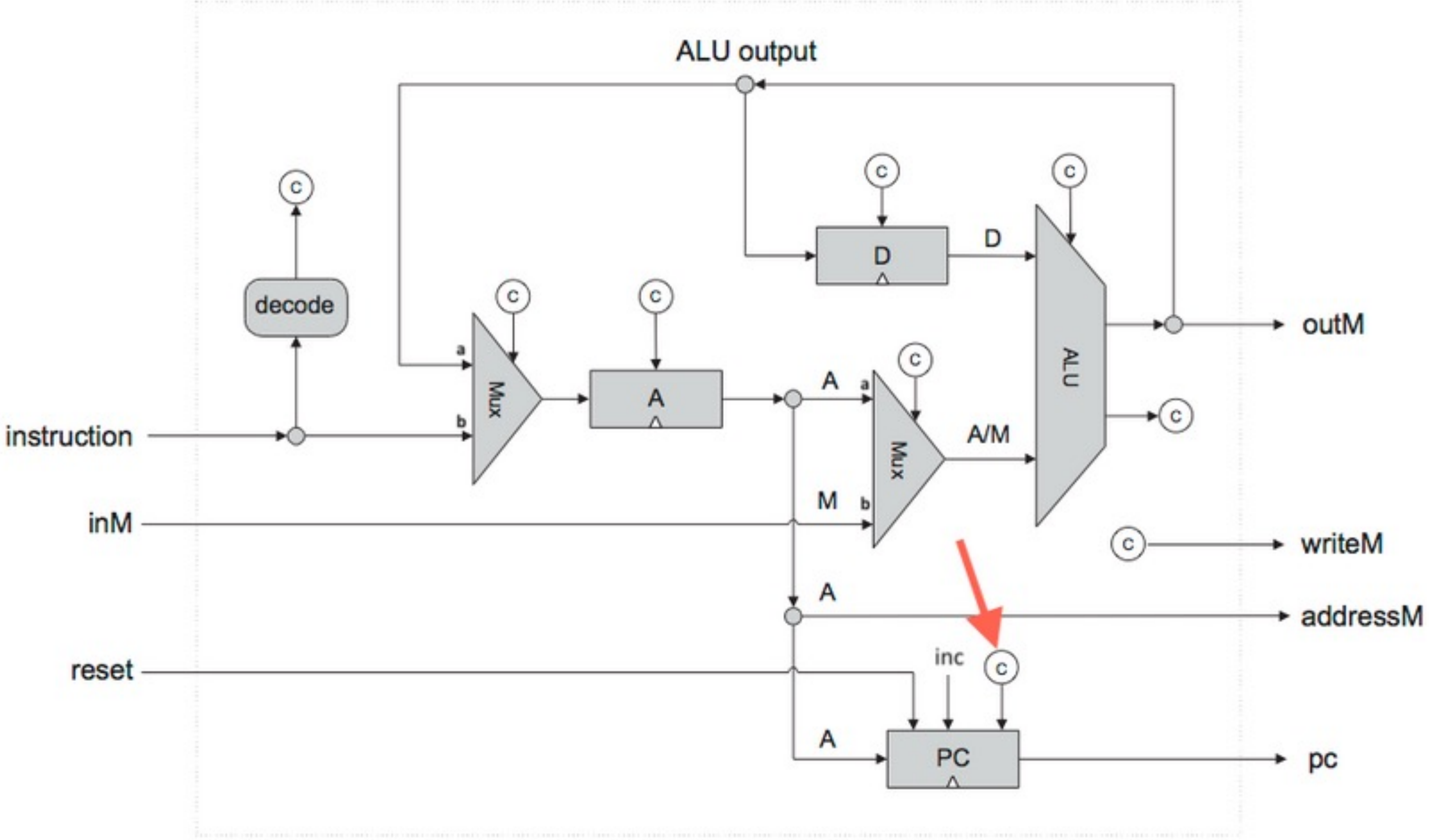

Figure 5.9 Proposed CPU implementation. The diagram shows only *data* and *address paths*, namely, wires that carry data and addresses from one place to another. The diagram does not show the CPU's *control logic*, except for inputs and outputs of control bits, labeled with a circled "c". Thus it should be viewed as an incomplete chip diagram.

THE UNIVERSITY of ADELAIDE

Question 3

1 pts

What does the following Hack assembler code always do to the current value in register D?

```
D=!D
D=D+1
```

- ◯ Sets D to be 0
- ◯ Sets D to be 1 - D
- ◯ Sets D to be 1
- ◯ Sets D to be -D

Question 4

Category: Assembler

Briefly describe what the following Hack assembler code does

```
        @KBD
        D=M
        @48
        D=D-A
        @num
        M=D
(END)
        @END
        0;JMP
```

- ◯ It reads the keboard register at the time of the @kbd command and exits if the scan code is more than 48
- ◯ It reads the keyboard code at the time of the first D=M command and subtracts 48 from it and puts the result in "num"
- ◯ Nothing, there is a syntax error and the code doesn't assemble.
- ◯ Nothing, there is no way that this code is able to read the keyboard unless the user is able to press the number '48' which doesn't exist on the keyboard. If it could work it would put the value -48 in "num".

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

THE UNIVERSITY of ADELAIDE

Question 5

1 pts

Category: Assembler

What does the following Hack assembler code do?

```
        @pix
        M=1
        D=M
        M=M+D
        M=M+1
        D=M
        M=M+D
        M=M+1
        D=M
        @SCREEN
        M=D
(END)
        @END
        0;JMP
```

- ○ It draws one pixel black.
- ○ It draws no pixels black because of overflow.
- ○ It draws the following pixels black, pixel 0, pixel 16 and pixel 32.
- ○ It draws the topmost leftmost three pixels on the screen black.

Question 5

1 pts

Category: Assembler

What does the following Hack assembler code do?

```
        @pix
        M=1
        D=M
        M=M+D
        M=M+1
        D=M
        M=M+D
        M=M+1
        D=M
        @SCREEN
        M=D
(END)
        @END
        0;JMP
```

A	D	M	@pix	@SCREEN
16	?	?	?	?
16384				

THE UNIVERSITY of ADELAIDE

This Week

- Review Chapters 5 & 6 of the textbook (if you haven’t already)
- Assignment 3 extension due this Sunday.
- Assignment 4 due this Sunday.
- Review Chapter 7 of the textbook before next week.

THE UNIVERSITY of ADELAIDE