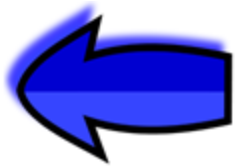
CS301 Computer Architecture

DR GAYATHRI ANANTHANARAYANAN

gayathri@iitdh.ac.in

Materials in these slides are borrowed from textbooks and existing Architecture courses

Outline

* Outline of a Processor
* Detailed Design of each Stage
* The Control Unit
* Microprogrammed Processor 
* Microassembly Language
* The Microcontrol Unit

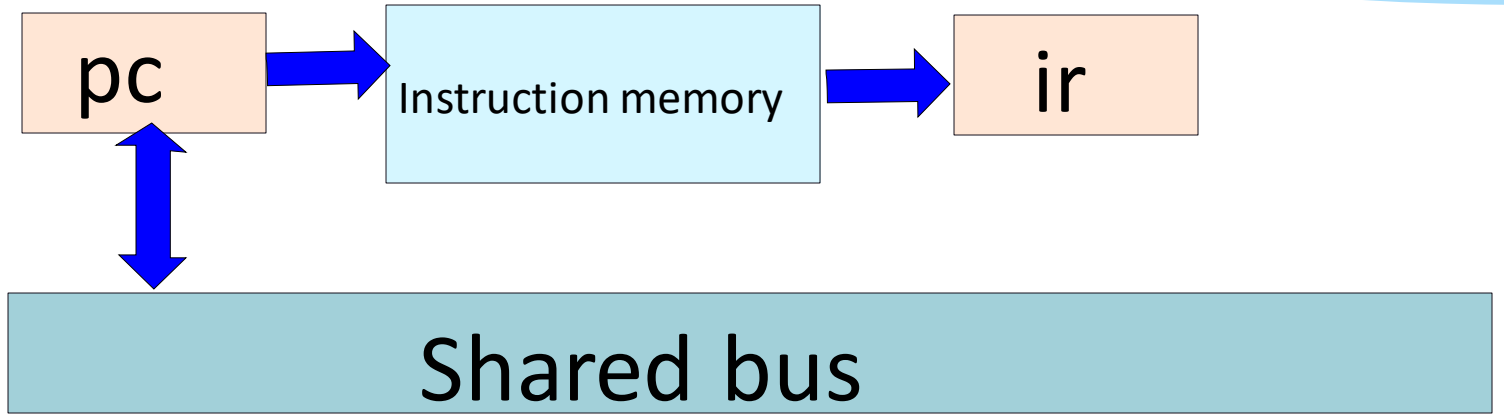
Microprogramming

* Idea of microprogramming
  * Expose the elements in a processor to software
  * Implement instructions as dedicated software routines
* Why make the implementation of instructions flexible ?
  * Dynamically change their behaviour
  * Fix bugs in implementations
  * Implement very complex instructions

Microprogrammed Data Path

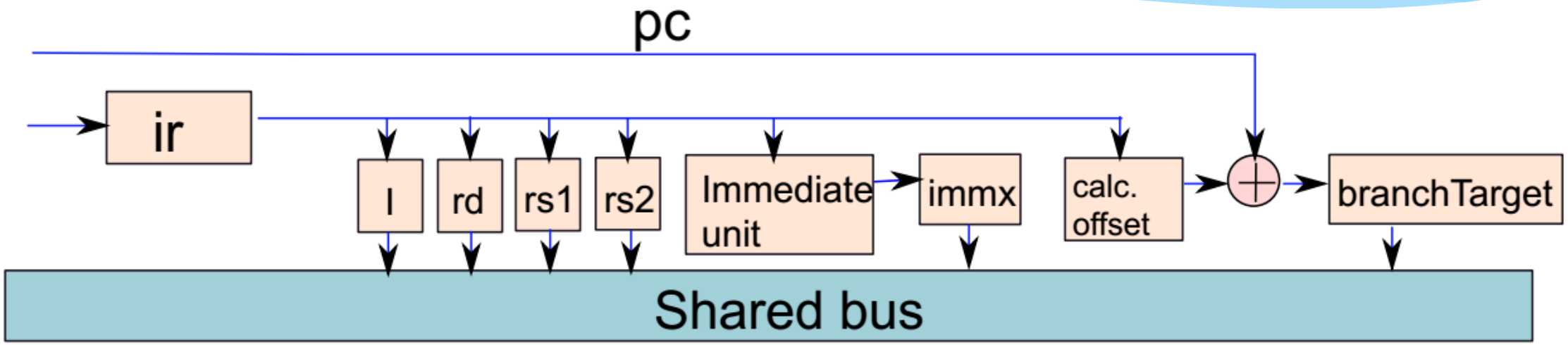
* Expose all the state elements to dedicated system software – firmware
* Write dedicated routines in firmware for implementing each instruction
* Basic idea
  * 1 SimpleRisc Instruction → Several micro instructions
  * Execute each micro instruction
  * We require a microprogram counter, and microinstruction memory

Fetch Unit



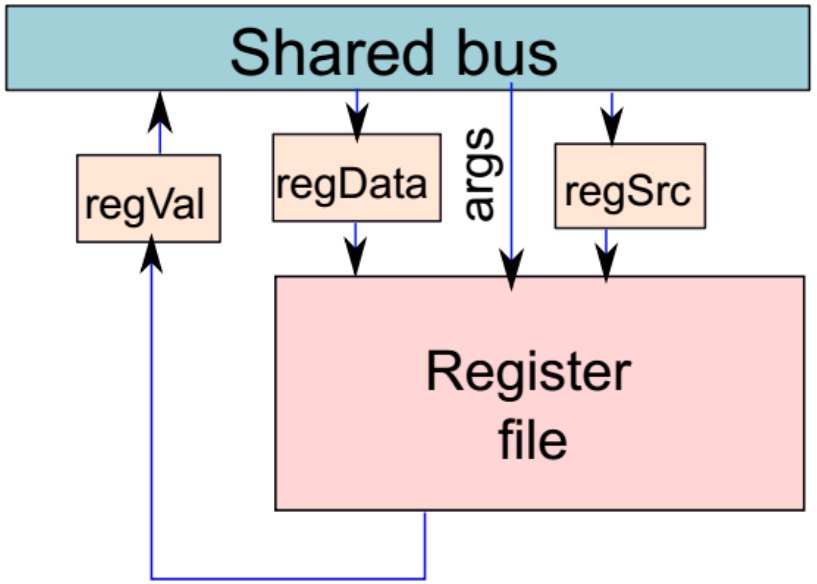
* The pc is used to access the instruction memory.
* The contents of the instruction are saved in the instruction register (ir)

Decode Unit



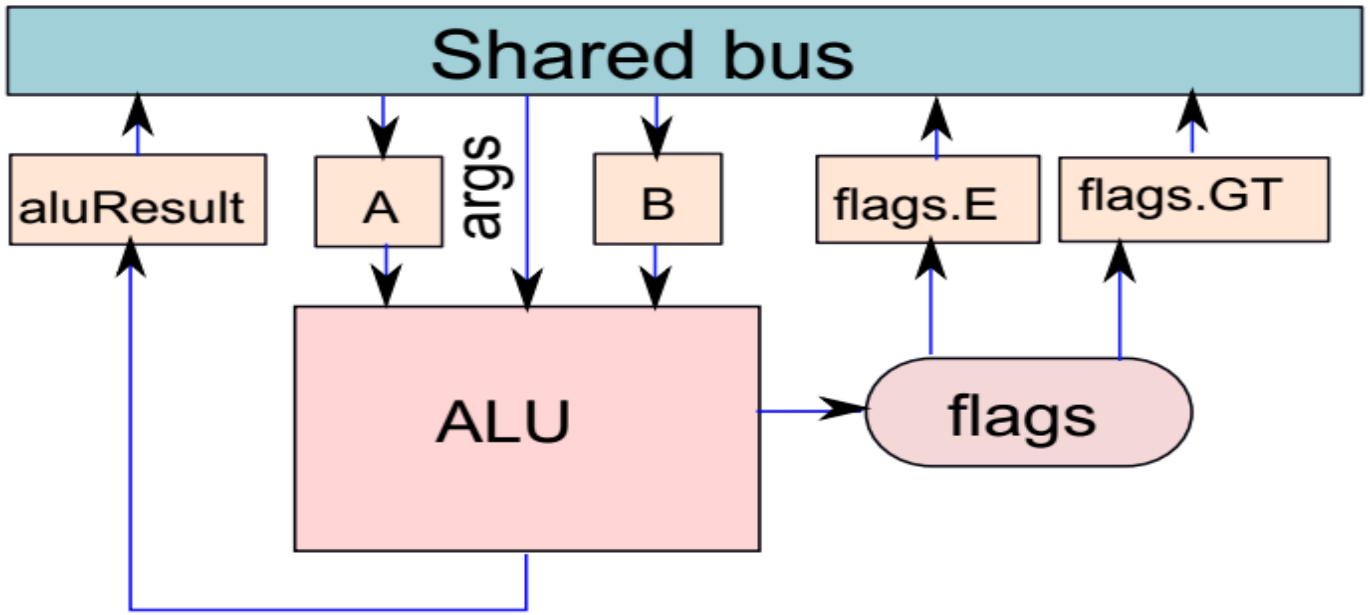
* Divide the contents of ir into different fields
  * I, rd, rs1, rs2, immx, and branchTarget

The Register File



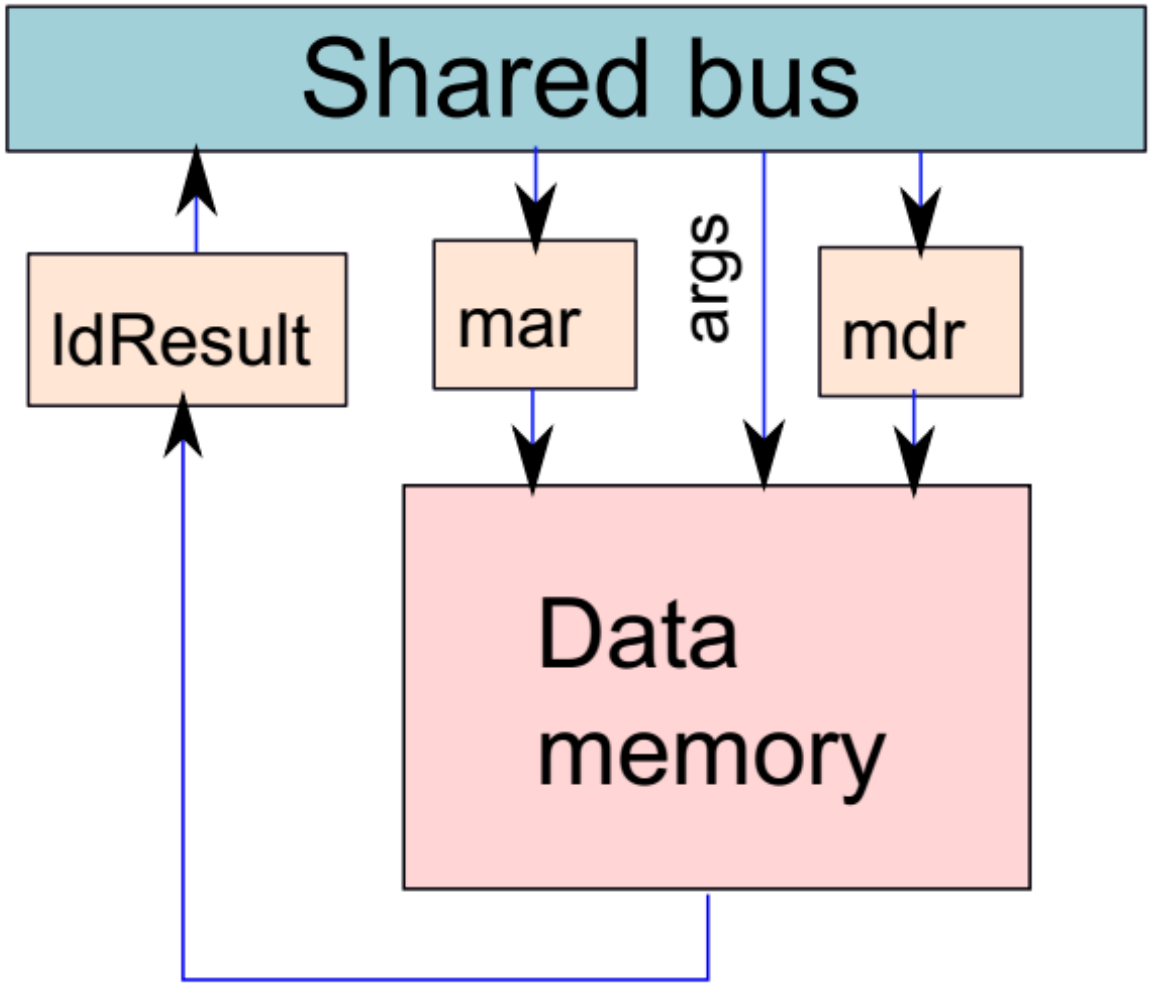
* regSrc (id of the source/dest register)
* regData (data to be stored)
* regVal (register value)

ALU

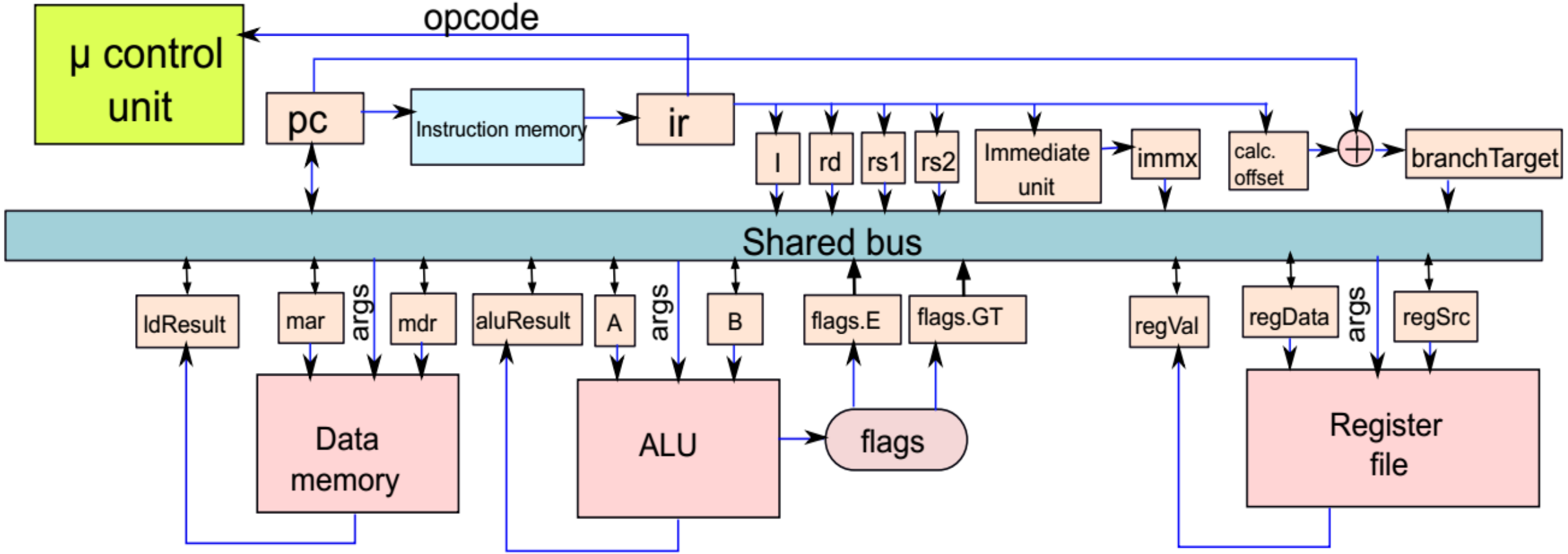


* A, B → ALU operands
* args → ALU operation type
* aluResult → ALU Result

Memory Unit



Microprogrammed Data Path



Microprogrammed control unit (μcontrol unit)

- Execute a set of microinstructions corresponding to each program instruction, and orchestrate the flow of data values across the different units in the data path of the microprogrammed processor.
- Responsible for the execution of microinstructions, data transfers across the different units, and for transferring control to the correct program instruction by updating the PC
- Added an extra connection between the ir register and the μcontrol unit to transfer the opcode of the instruction.
- Require the μcontrol unit to load the appropriate set of microinstructions corresponding to the program instruction. By design, we do not wish to make the value of the opcode available to other units.
- We have a set of microinstructions for each opcode, and there is no reason why other units should require the value of the opcode.

Internal Registers

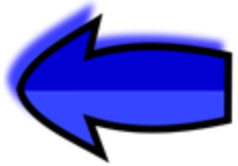
SerialNo.	Register	Size (bits)	Function
1	*pc*	32	program counter
2	*ir*	32	instruction register
3	*I*	1	immediate bit in the instruction
4	*r*	4	destination register id
5	*rs* 1	4	id of the first source register
6	*rs* 2	4	id of the second source register
7	*immx*	32	immediate embedded in the instruction (after processing modifiers)
8	*branchTarget*	32	branch target, computed as the sum of the PC and the offset embedded in the instruction
9	*regSrc*	4	contains the id of the register that needs to be accessed in the register file
10	*regData*	32	contains the data to be written into the register file

Internal Registers - II

11	*regVal*	32	value read from the register file
12	*A*	32	first operand of the AlU
13	*B*	32	second operand of the ALU
14	*flags.E*	1	the equality flag
15	*flags.GT*	1	the greater than flag
16	*aluResult*	32	the ALU result
17	*mar*	32	memory address register
18	*mdr*	32	memory data register
19	*ldResult*	32	the value loaded from memory

Outline

* Outline of a Processor
* Detailed Design of each Stage
* The Control Unit
* Microprogrammed Processor
* Microassembly Language
* The Microcontrol Unit



Microinstructions

Basic Instructions

* mloadIR → Loads the instruction register (ir) with the contents of the instruction.
* mdecode → Waits for 1 cycle. Meanwhile, all the decode registers get populated
* mswitch → Loads the set of micro instructions corresponding to a program instruction.

Move Microinstructions

* mmov r1, r2 : r1 ← r2
* mmov r1, r2, <args> : r1 ← r2, send the value of args on the bus
* mmovi r1, <imm> : r1 ← imm

Add and Branch Microinstructions

* madd r1, imm, <args>
  * r1 ← r1 + imm
  * send <args> on the bus
* mbeq r1, imm, <label>
  * if (r1 == imm), μpc = addr(label)
* mb <label>
  * μpc = addr(label)

Add and Branch Microinstructions

Microinstruction	Semantics
$madd\ r1, < imm >$	$r1 \leftarrow r1 + imm$
$madd\ r1, < imm >, < args >$	$r1 \leftarrow r1 + imm$, send the value of $< args >$ on the bus

Microinstruction	Semantics
$mb < addr >$	execute the microinstruction at $< addr >$(label) in the microprogram memory
$mbeq\ reg, imm, < addr >$	If the value in the internal register reg is equal to imm then the microPC needs to jump to $< addr >$(label)

Summary of Microinstructions

SerialNo.	Microinstruction	Semantics
1	*mloadIR*	*ir* ← [*pc*]
2	*mdecode*	populate all the decode registers
3	*mswitch*	jump to the *μpc* corresponding to the opcode
4	*mmov reg*1, *reg*2, < *args* >	*reg*1 ← *reg*2, send the value of *args* to the unit that owns *reg*1, < *args* > is optional
5	*mmovi reg*1, *imm*, < *args* >	*reg*1 ← *imm*, < *args* > is optional
6	*madd reg*1, *imm*, < *args* >	*reg*1 ← *reg*1+*imm*, < *args* > is optional
7	*mbeq reg*1, *imm*, < *label* >	if (*reg*1 = *imm*) *μpc* ← *addr*(*label*)
8	*mb* <*label*>	*μpc* ← *addr*(*label*)

Implementing Instructions in Microcode

* The microcode preamble

```
.begin:
mloadIR
mdecode
madd pc, 4
mswitch
```

  * Load the program counter
  * Decode the instruction
  * Add 4 to the pc
  * Switch to the first microinstruction in the microcode sequence of the prog. instruction

```
1 mmov regSrc, rs1, <read>
2 mmov A, regVal
```

```
1 mbeq I, 1, .imm
2 /* second operand is a register */
3 mmov regSrc, rs2, <read>
4 mmov B, regVal, <aluop>
5 mb .rw
6 /* second operand is an immediate */
7 .imm:
8 mmov B, immx, <aluop>
9 /* write the ALU result to the register file*/
10 .rw:
```

```
1 .rw:
2         mmov regSrc, rd
3         mmov regData, aluResult, <write>
4         mb .begin
```

3 Address Format ALU Instruction

```
/* transfer the first operand to the ALU */
mmov regSrc, rs1, <read>
mmov A, regVal

/* check the value of the immediate register */
mbeq I, 1, .imm
/* second operand is a register */
mmov regSrc, rs2, <read>
mmov B, regVal, <aluop>
mb .rw
/* second operand is an immediate */
.imm:
mmov B, immx, <aluop>

/* write the ALU result to the register file*/
.rw:
mmov regSrc, rd
mmov regData, aluResult, <write>
mb .begin
```

The *mov* Instruction

mov instruction

```
1  /* check the value of the immediate register */
2  mbeq I, 1, .imm
3  /* second operand is a register */
4  mmov regSrc, rs2, <read>
5  mmov regData, regVal
6  mb .rw
7  /* second operand is an immediate */
8  .imm:
9  mmov regData, immx
10
11 /* write to the register file*/
12 .rw:
13 mmov regSrc, rd, <write>
14
15 /* jump to the beginning */
16 mb .begin
```

The *not* Instruction

not instruction

```
1  /* check the value of the immediate register */
2  mbeq I, 1, .imm
3  /* second operand is a register */
4  mmov regSrc, rs2, <read>
5  mmov B, regVal, <not> /* ALU operation */
6  mb .rw
7  /* second operand is an immediate */
8  .imm:
9  mmov B, immx, <not> /* ALU operation */
10
11 /* write to the register file*/
12 .rw:
13 mmov regData, aluResult
14 mmov regSrc, rd, <write>
15
16 /* jump to the beginning */
17 mb .begin
```

The *cmp* Instruction

cmp instruction –

```
1  /* transfer rs1 to register A */
2  mmov regSrc, rs1, <read>
3  mmov A, regVal
4
5  /* check the value of the immediate register */
6  mbeq I, 1, .imm
7  /* second operand is a register */
8  mmov regSrc, rs2, <read>
9  mmov B, regVal, <cmp> /* ALU operation */
10 mb .begin
11
12 /* second operand is an immediate */
13 .imm:
14 mmov B, immx, <cmp> /* ALU operation */
15 mb .begin
```

The *nop* Instruction

* mb .begin

The *ld* Instruction

ld instruction

```
1  /* transfer rs1 to register A */
2  mmov regSrc, rs1, <read>
3  mmov A, regVal
4
5  /* calculate the effective address */
6  mmov B, immx, <add> /* ALU operation */
7
8  /* perform the load */
9  mmov mar, aluResult, <load>
10
11 /* write the loaded value to the register file */
12 mmov regData, ldResult
13 mmov regSrc, rd, <write>
14
15 /* jump to the beginning */
16 mb .begin
```

The *st* Instruction

st instruction

```
1  /* transfer rs1 to register A */
2  mmov regSrc, rs1, <read>
3  mmov A, regVal
4
5  /* calculate the effective address */
6  mmov B, immx, <add> /* ALU operation */
7
8  /* perform the store */
9  mmov mar, aluResult
10 mmov regSrc, rd, <read>
11 mmov mdr, regVal, <store>
12
13 /* jump to the beginning */
14 mb .begin
```