b instruction

```
mmov pc, branchTarget
mb .begin
```

beq instruction

```
/* test the flags register */
mbeq  flags.E, 1, .branch
mb. begin

.branch:
mmov pc, branchTarget
mb .begin
```

bgt instruction

```
/* test the flags register */
mbeq  flags.GT, 1, .branch
mb. begin

.branch:
mmov pc, branchTarget
mb .begin
```

beq and *bgt* Instructions

beq instruction

```
/* test the flags register
mbeq flags.E, 1, .branch
mb .begin

.branch:
mmov pc, branchTarget
mb .begin
```

bgt instruction

```
/* test the flags register
mbeq flags.GT, 1, .branch
mb .begin

.branch:
mmov pc, branchTarget
mb .begin
```

call Instruction

call instruction

```
/* save PC + 4 in the return address register */
mmov regData, pc
mmovi regSrc, 15, <write>

/* branch to the function */
mmov pc, branchTarget
mb .begin
```

ret Instruction

ret instruction

```

/* save the contents of the return
                 address register in the PC */
mmovi regSrc, 15, <read>
mmov pc, regVal
mb .begin
```

Example

Change the call instruction to store the return address on the stack. The preamble need not be shown.
Answer:

```
              stack based call instruction
/* read the stack pointer */
mmovi regSrc, 14, <read>
madd regVal, -4  /* decrement the stack pointer */

/* set the memory address to the stack pointer *
/ * MAR contains the new stack pointer */
mmov mar, regVal

/* update the stack pointer */
mmov regData, regVal, <write> /* update stack pointer */

/* write the return address to the stack */
mmov mdr, pc, <store>

/* jump to the beginning */
mb .begin
```

Change the ret instruction to load the return address from the stack. The preamble need not be shown.

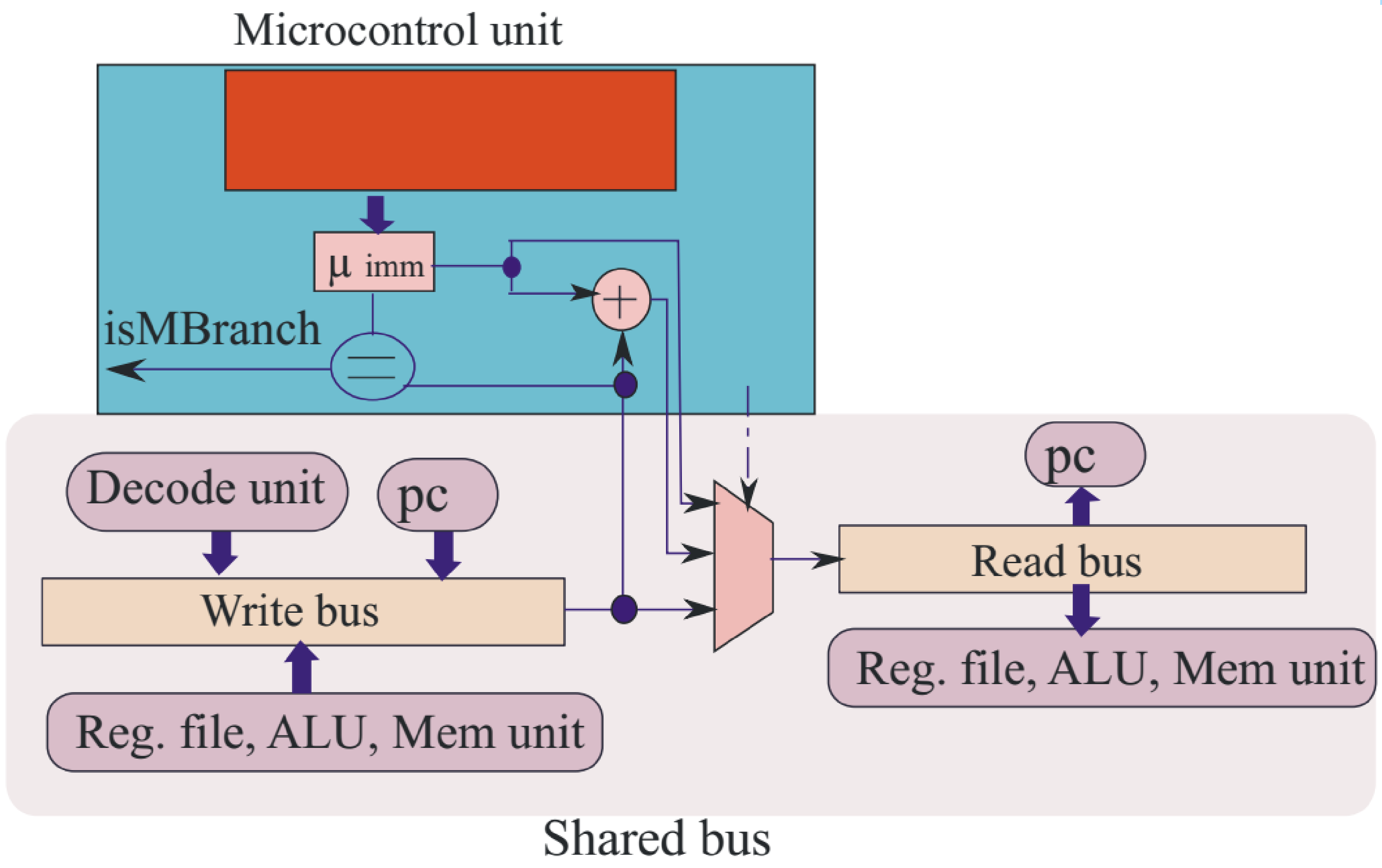
stack based call instruction

```
/* read the stack pointer */
mmovi regSrc, 14, <read>

/* set the memory address to the stack pointer */
mmov mar, regVal, <load>

mmov pc, ldResult /* set the PC */

/* update the stack pointer */
madd regVal, 4 /* sp = sp + 4 */
mmov regData, regVal, <write> /* update stack pointer */

/* jump to the beginning */
mb .begin
```

Outline

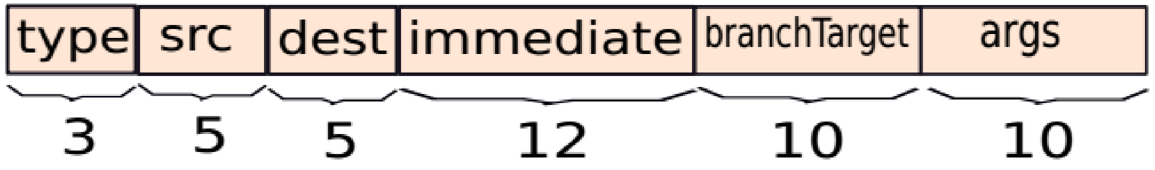
- Outline of a Processor
- Detailed Design of each Stage
- The Control Unit
- Microprogrammed Processor
- Microassembly Language
- The Microcontrol Unit

Shared Bus


Microcontrol unit
μ imm
isMBranch
+
=
Decode unit
pc
Write bus
Reg. file, ALU, Mem unit
pc
Read bus
Reg. file, ALU, Mem unit
Shared bus

Encoding an Instruction

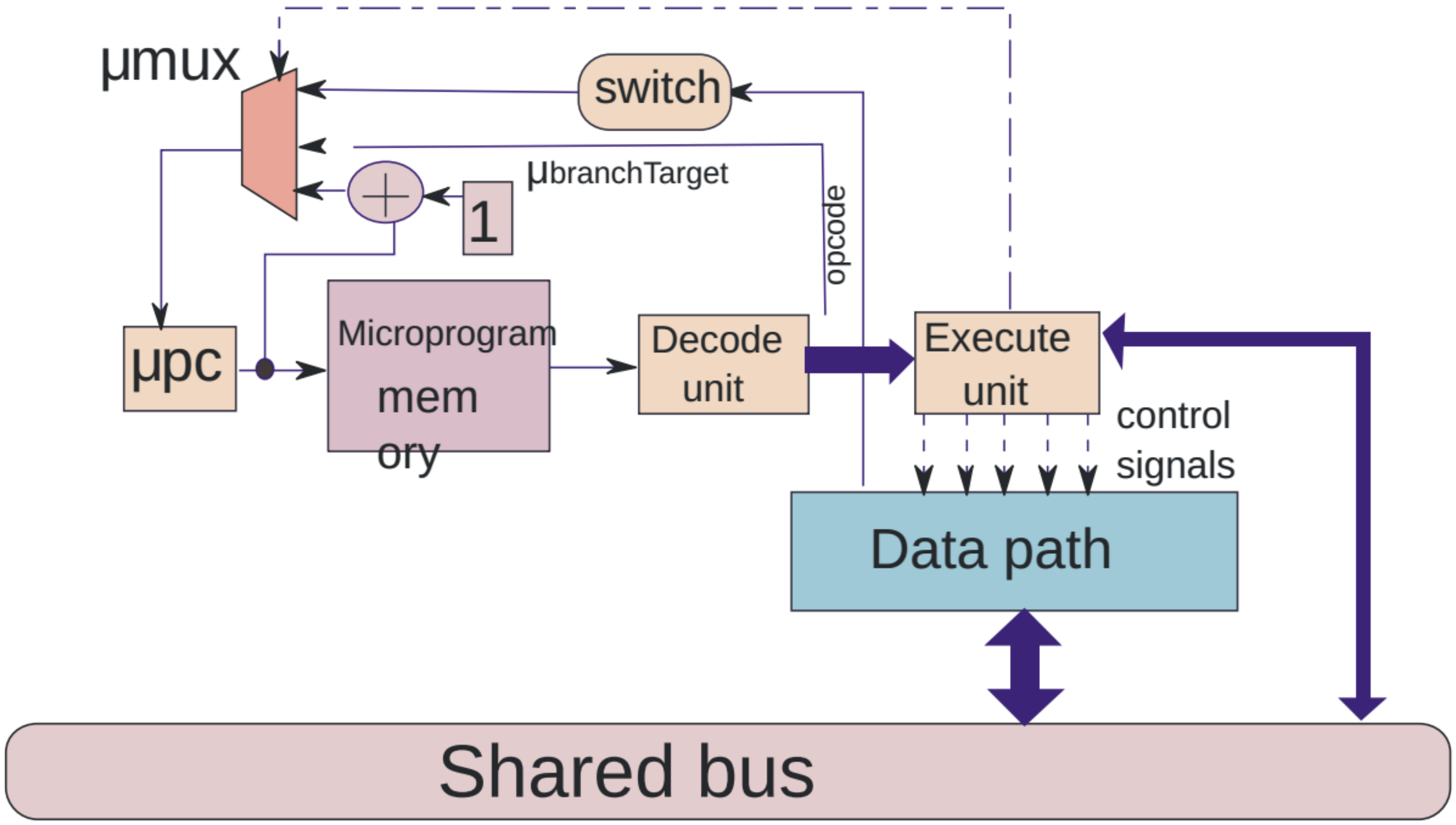
- Vertical Microprogramming (45 bit inst.)
  - 3 bits → type of instruction
  - 5 bits → source register
  - 5 bits → destination register
  - 12 bits → immediate
  - 10 bit → branch target in microcode memory
  - 10 bit → args value
    - 3 bits → (unit id)
    - 7 bits → operation code



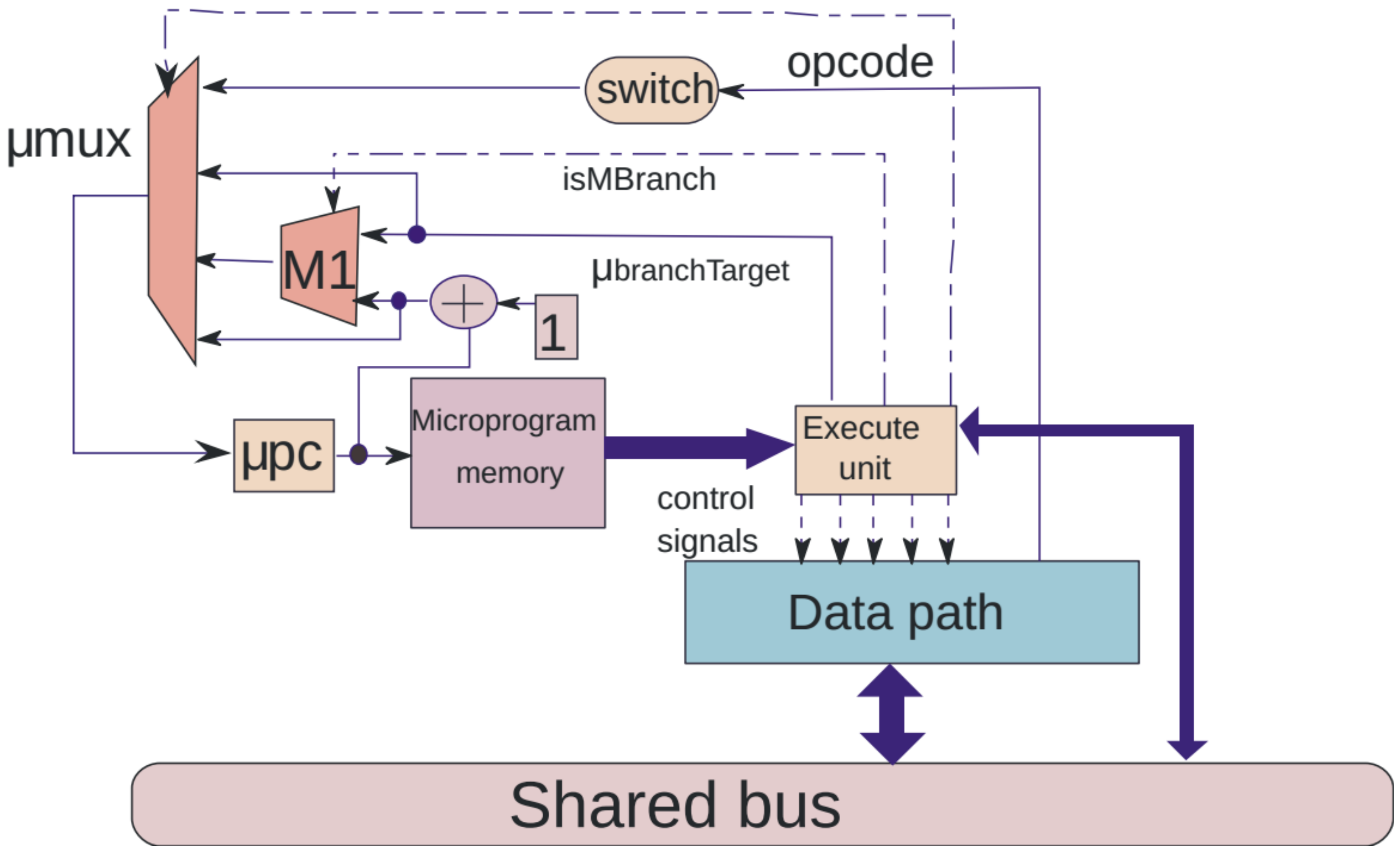
Horizontal Microprogramming

- Encoding
  - 10 bits → branch target
  - 12 bits → immediate
  - 10 bits → args
  - 33 bits → bit vector of all the control signals
- Total size of the encoded instruction : 65 bits

Vertical Microprogramming



Horizontal Microprogramming


opcode
switch
μmux
isMBranch
M1
μbranchTarget
1
Microprogram memory
μpc
Execute unit
control signals
Data path
Shared bus