MIPS on FLEET

Amir Kamil

Goals

- Run most MIPS assembly code on FLEET
  - Attempt to duplicate level of support in SPIM interpreter
  - MIPS assembly translated to FLEET assembly
  - Small set of SHIPs used to implement MIPS operations
  - Register file abstraction provided
- Eventual goal: run C code on FLEET
  - Compile C → MIPS → FLEET

MIPS Instructions

- Each instruction is implemented as a codebag
- When an instruction is done, it sends the next instruction codebag to the FetchShip
- The next instruction’s release is predicated on notification from the previous instruction that it is done

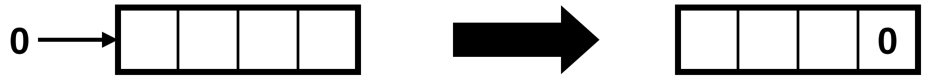
Instruction Example

- Example: `AND $t0, $s0, $s1`

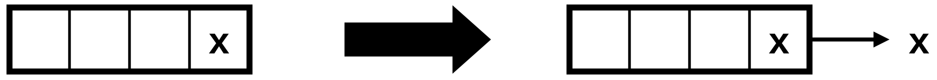
```
PC0x400000: {
  copy        s0fifo.out → logic.A
  copy        s1fifo.out → logic.B
              “AND”      → logic.cmd
  move        token.out  → logic.out
  move        logic.out  → t0fifo.in
  move        t0fifo.out → ()
  accept+ack  t0fifo.in  → fetch.release
              PC0x400400 → fetch.codebag
  move        fetch.done → ()
}
```

MIPS Registers

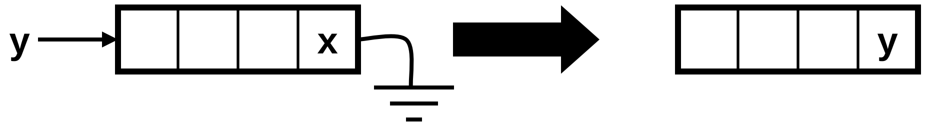
- Registers implemented using FIFOs
- Operations
  - Initialization: add 0 to FIFO



  - Read: copy output



  - Write: move input to FIFO, send output to bitbucket



MIPS ALU Operations

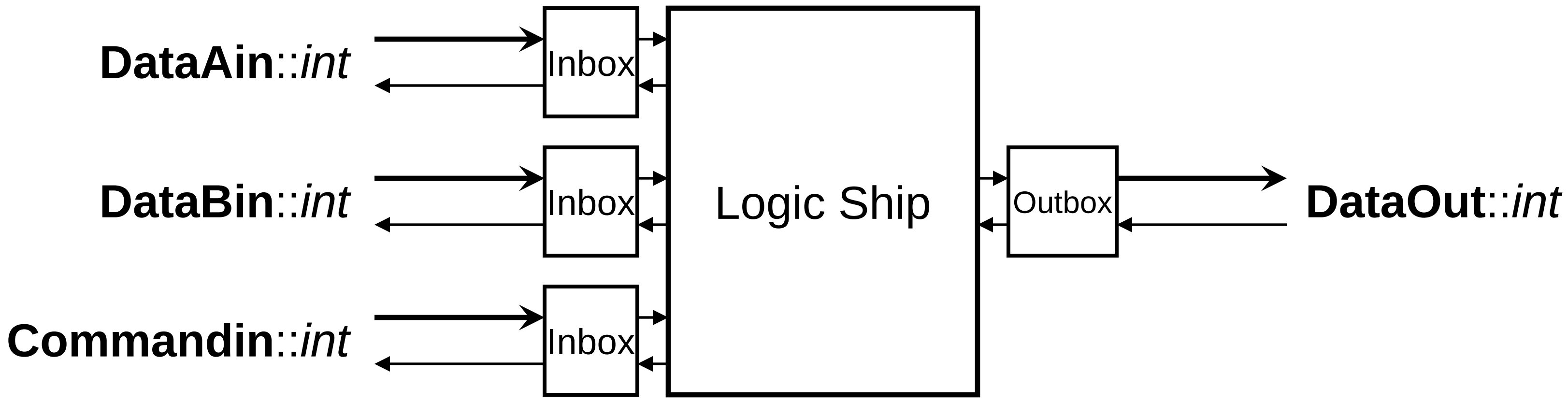
- ALU operations
  - Addition (`ADD`, `ADDI`, `ADDIU`, `ADDU`)
  - Subtraction (`SUB`, `SUBU`)
  - Multiplication (`MULT`, `MULTU`)
  - Division (`DIV`, `DIVU`)
  - Shifts (`SLL`, `SLLV`, `SRA`, `SRAV`, `SRL`, `SRLV`)
  - Logic (`AND`, `ANDI`, `OR`, `ORI`, `XOR`, `XORI`, `NOR`)

Arithmetic Implementation

- Addition and subtraction implemented directly using ArithmeticShip
- Signed multiplication uses Dominic’s MultiplierShip
- Unclear what to do for division and unsigned multiplication
  - Can build on top of ArithmeticShip, MultiplerShip, and ShiftShip, but result complicated and slow
  - Can also build into hardware, on top of MIPS instructions, or a combination of the above

Logic Implementation

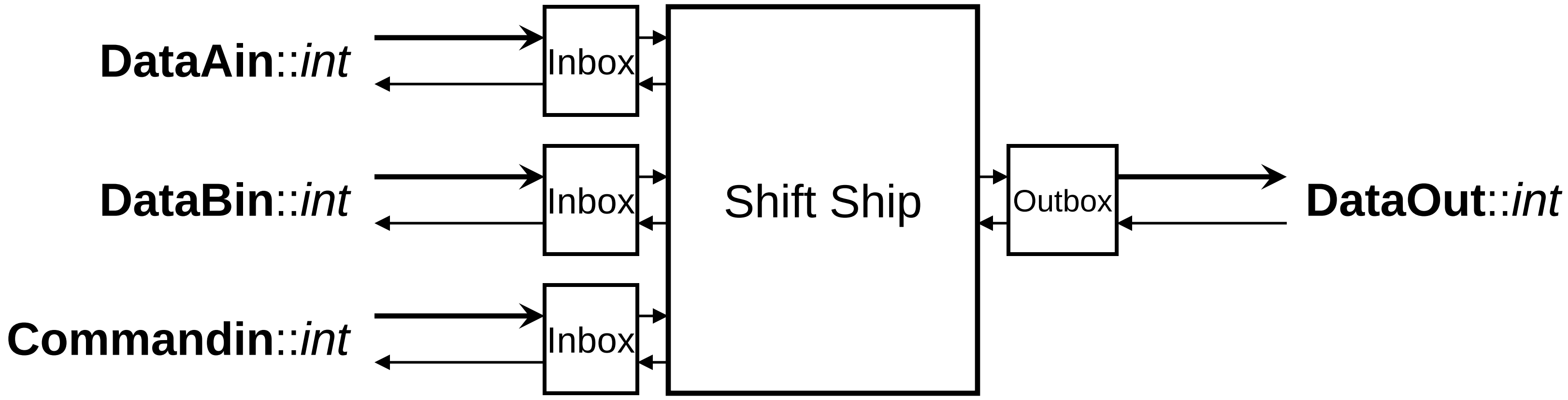
- Logical operations can be built either on top of the BitwiseShip, or on a simpler LogicShip
  - Since no one has built a BitwiseShip, I'm using a LogicShip for now



  - Commands: AND, OR, XOR, NOR

Shift Implementation

- Shift operations can use the IES31 ShiftShip
  - However, lots of operations required to perform a shift of more than 1
  - Also, complicated to implement variable shifts
  - Can use new ship:



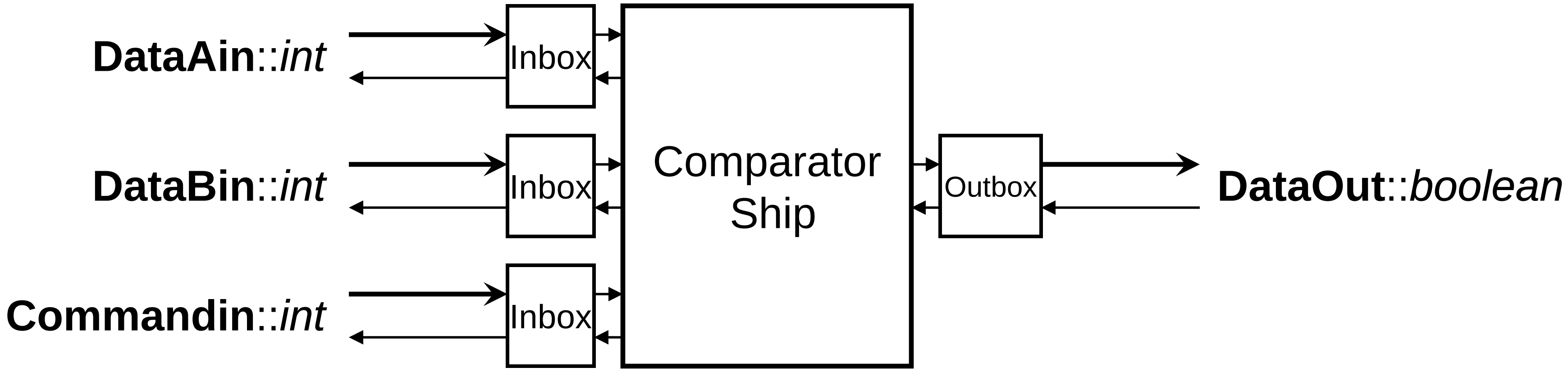
  - Commands: SLL, SRA, SRL

MIPS Control Flow Operations

- Control flow operations
  - Branches (`BEQ`, `BGEZ`, `BGTZ`, `BLEZ`, `BLTZ`, `BNE`)
  - Jumps (`J`, `JAL`, `JALR`, `JR`)
- Set operations
  - Set on less than (`SLT`, `SLTI`, `SLTIU`, `SLTU`)

Comparisons

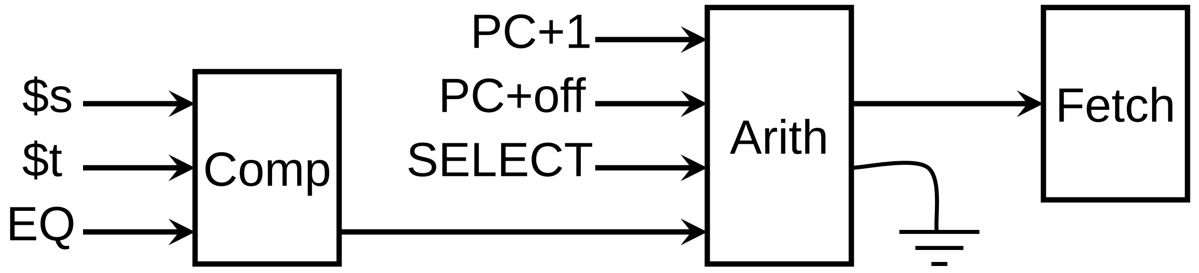
- Branches and sets compare values, so a ComparatorShip needs to be defined
  - Can use ArithmeticShip, but slower and more complicated



  - Commands: EQ, NEQ, GT, GEQ, LT, LEQ

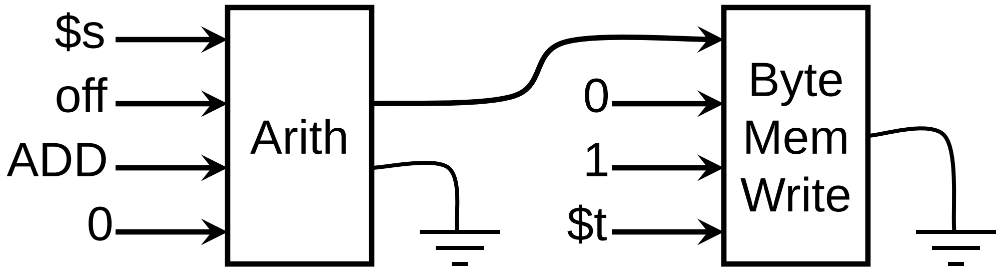
Selections

- Branches and sets also select between two values
  - The SELECT command on the ArithmeticShip can be used to do so
  - Example: `BEQ $s, $t, off`



MIPS Memory Access Operations

- Memory access operations
  - Loads (`LB`, `LBU`, `LUI`, `LH`, `LHU`, `LW`)
  - Stores (`SB`, `SH`, `SW`)
- Memory byte-addressed, so need new byte-addressed memory ships
  - Example: `SW $t, off($s)`



Other MIPS Operations

- Register moves (`MFHI`, `MTHI`, `MFLO`, `MTLO`)
  - Easy to implement
- System calls (`SYSCALL`)
  - Only a few calls implemented
- Floating point instructions
  - Not implemented

Future Work

- Add more complete support for system calls
  - Need I/O specification for FLEET first
- Add floating point instructions
  - Need floating point specification for FLEET first
- Optimize instruction sequencing logic
  - Remove sequencing where it is unnecessary
- Add support for instruction-level parallelism
  - Take advantage of duplicated SHIPs, or even multiple FLEETs