05 - Arithmetic Logic Unit

Andreas Ehliar

September 16, 2014

Special Purpose Registers (continued from last lecture)

- Sometimes we need special purpose registers (SPR or SR)
  - BOT/TOP for modulo addressing
  - AR for address register
  - SP
  - I/O
  - Core configuration registers
  - etc
- Should these be included in the general purpose register file?

Special Purpose Registers as normal registers

- Convenient for the programmer. Special purpose registers can be accessed like any normal register.
  - Example: `add bot0,1 ; Move ringbuffer bottom one word`
  - Example 2 (from ARM): `pop pc`
- Drawbacks:
  - Wastes entries in the general purpose register file
  - Harder to use specialized register file memories

Special purpose registers needs special instructions

- Special instructions required to access SR:s
- Example:
  - `move r0,bot0` ; Move ringbuffer bottom one word
  - `(nop) ; May need nop(s) here`
  - `add r0,1`
  - `(nop) ; May need nop(s) here`
  - `bot0,r0`
  - (Move is encoded as move from from/to special purpose register here)
- Advantage:
  - Easier to meet timing as special purpose registers can easier be located anywhere in the core
  - Can scale easily to hundreds of special purpose registers if required. (Common on large and complex processors such as ARM/x86)
- Drawback:
  - Inconvenient for special registers you need to access all the time

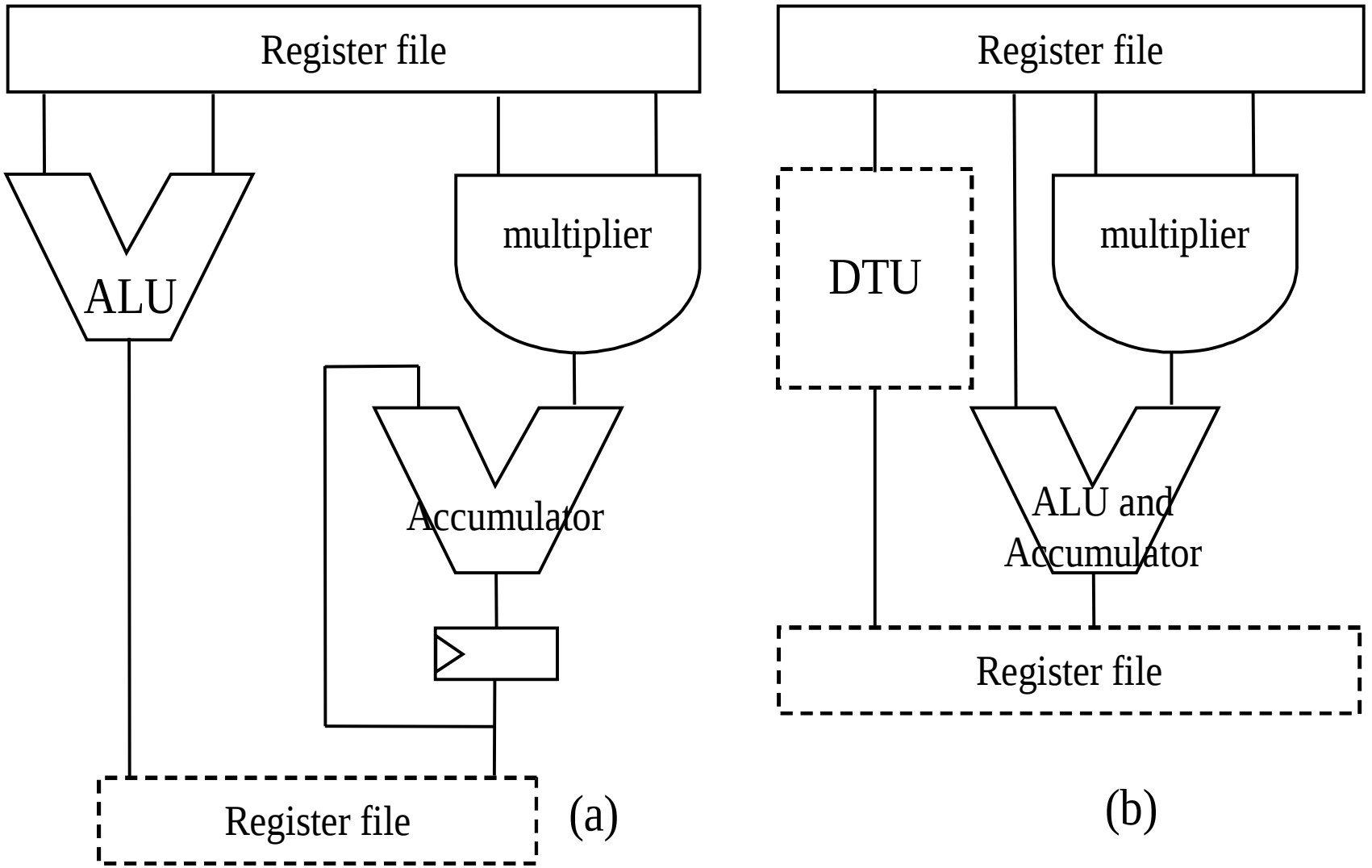
Conclusions: SPRs

- Only place SPRs as a normal register if you believe it will be read/written via normal instructions very often

ALU in general

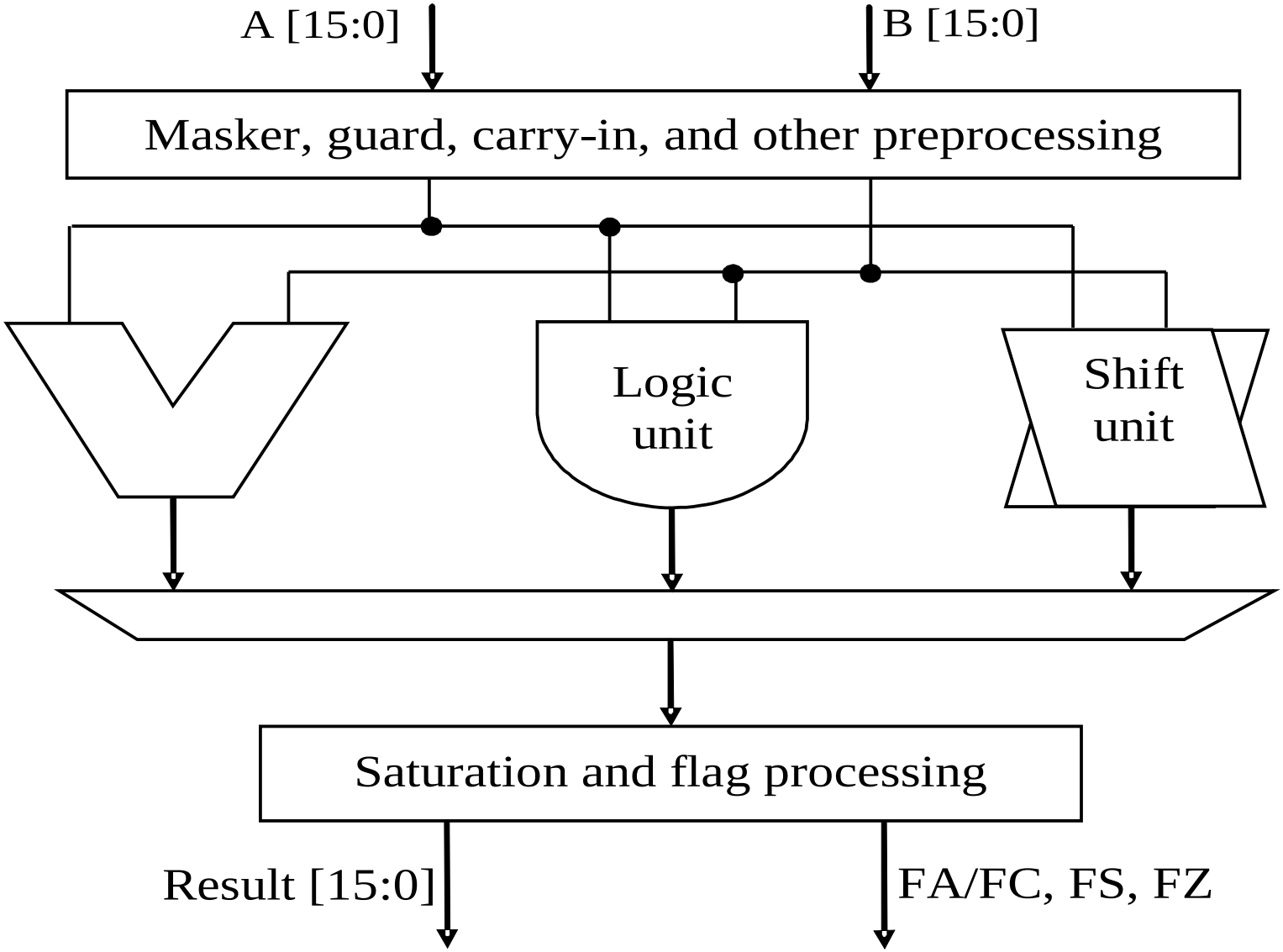
- ALU: Arithmetic and Logic Unit
  - Arithmetic, Logic, Shift/rotate, others
  - No guard bits for iterative computing
  - One guard bit for single step computing
  - Get operands from and send result to RF
  - Handles single precision computing

Separate ALU or ALU in MAC



[Liu2008]

ALU high level schematic



[Liu2008]

Pre-processing

- Select operands: from one of the source
  - Register file, control path, HW constant
- Typical operand pre processing:
  - Guard: one guard
    - (does not support iterative computing)
  - Invert: Conditional/non-conditional invert
  - Supply constant 0, 1, -1
  - Mask operand(s)
  - Select proper carry input

Post-processing

- Select result from multiple components
  - From AU, logic unit, shift unit, and others
- Saturation operation
  - Decide to generate carry-out flag or saturation
  - Perform saturation on result if required
- Flag operation
  - Flag computing and prediction

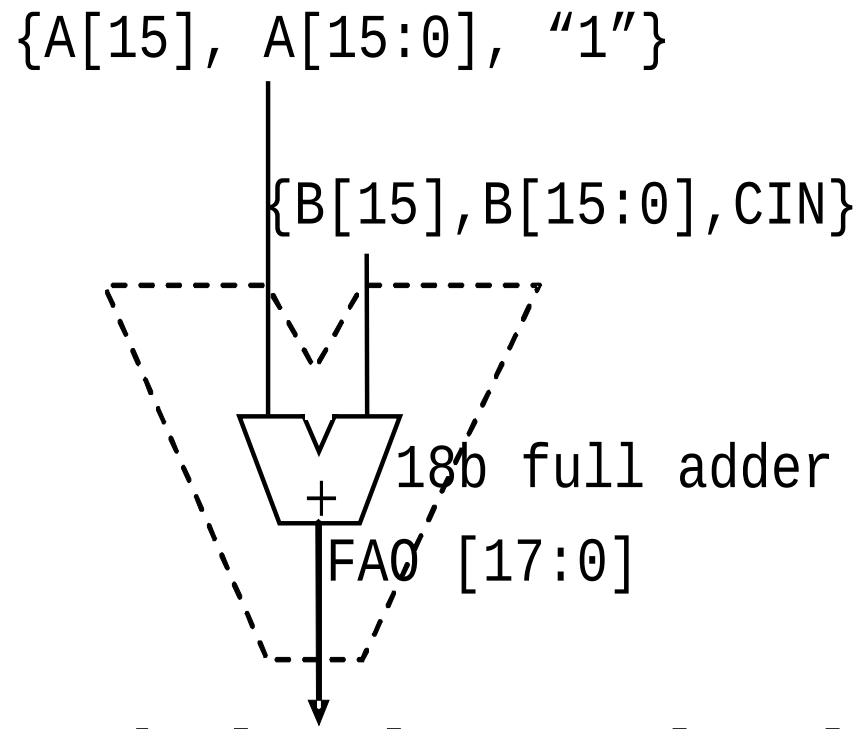
General instructions

	Operation	opa	opb	Carry in	Carry out
ADD	Addition	+	+	0	Cout/SAT
SUB	Subtraction	+	-	1	Cout/SAT
ABS	Absolute	+/-		A[15]	SAT
CMP	Compare	+	-	1	SAT
NEG	Negate	-		1	SAT
INC	Increment	+	1	0	SAT
DEC	Decrement	+	-1	0	SAT
AVG	Average	+	+	0	SAT

Special Instructions

Mnemonic	Description	Operation
MAX	Select larger value	RF <= max(OpA,OpB)
MIN	Select smaller value	RF <= min(OpA,OpB)
DTA	Difference of two absolute values	GR <= |OpA| − |OpB|
ADT	Absolute of the difference of two values	GR <= |OpA−OpB|

Adder with carry in for RTL synthesis (safe solution)



[Liu2008]

- Full adder may have no carry in
- One guard bit
- We need 2 extra bits in the adder
- LSB of the 18b result will not be used
- MSB of the 18b result will be the guard
- Works on all synthesis tools

Adder for RTL synthesis (modern version)

- `Cout,Res[15:0]` = 1'b0,A[15:0]+1'b0,B[15:0]+Cin;
- `Cout` is 1 bit wide
- **Important:** `Cin` **is 1 bit wide!**
- Modern synthesis tools can usually handle this case without creating two adders
  - (I've had to resort to the "safe" version shown on the previous slide in a few cases though. For example when combining an adder with other logic in an FPGA.)

Example: Implement an 8 bit ALU

Instructions	Function	OP
NOP	No change of flags	0
A+B	$A + B$ (without saturation)	1
A-B	$A - B$ (without saturation)	2
SAT(A+B)	$A + B$ (with saturation)	3
SAT(A-B)	$A - B$ (with saturation)	4
SAT(ABS(A))	$\|A\|$ (absolute operation, saturation)	5
SAT(ABS(A+B))	$\|A + B\|$ (absolute operation, saturation)	6
SAT(ABS(A-B))	$\|A - B\|$ (absolute operation, saturation)	7
CLR S	Clear S flag (other flags unchanged)	8

- There shold be a negative, zero, and saturation flag!

Example: Implement an 8 bit ALU

Instructions	Function	OP
NOP	No change of flags	0
A+B	$A + B$ (without saturation)	1
A-B	$A - B$ (without saturation)	2
SAT(A+B)	$A + B$ (with saturation)	3
SAT(A-B)	$A - B$ (with saturation)	4
SAT(ABS(A))	$\|A\|$ (absolute operation, saturation)	5
SAT(ABS(A+B))	$\|A + B\|$ (absolute operation, saturation)	6
SAT(ABS(A-B))	$\|A - B\|$ (absolute operation, saturation)	7
CLR S	Clear S flag (other flags unchanged)	8

- There shold be a negative, zero, and saturation flag!
- Discussion topic: How many adders are needed for each operation?
- Discussion topic: How many guard bits are needed for each operation?

Problem: The following function must execute in < 2048 cycles

- Problem: The following function must execute in less than 2048 cycles

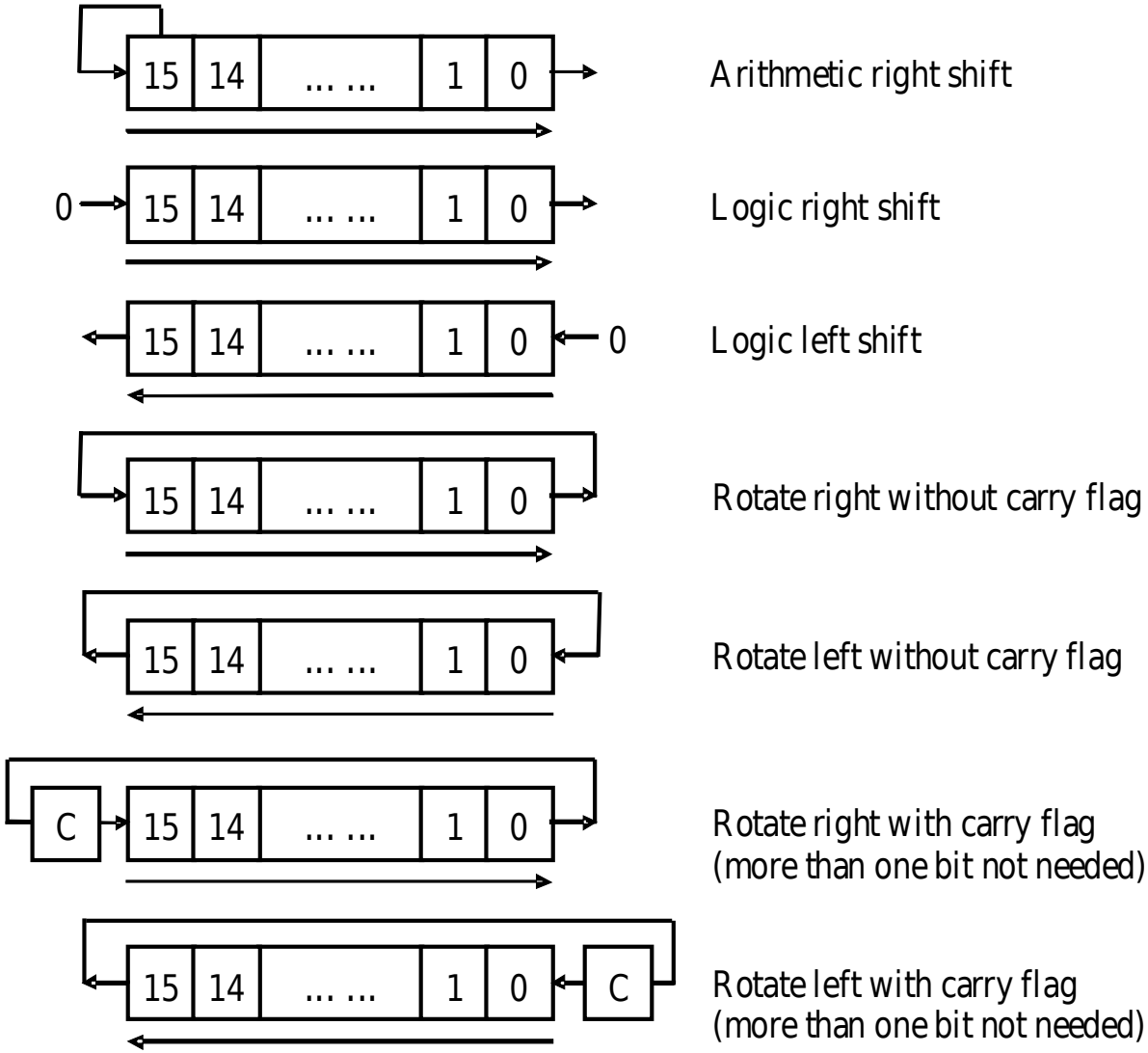
```
for(i=0; i<500;i++){
    tmp=abs(*p0++]);
    tmp2=abs(*p1++);
    *p2++ = tmp-tmp2;
}
```

```
; First try in assembler
    repeat 500,lend
    ld r0,DM0[ar0++]
    ld r1,DM0[ar1++]
    abs r0
    abs r1
    sub r0,r0,r1
    st DM0[ar2++],r0
lend:
; 3000 cycles for inner
; loop
```

16 bit $|A| - |B|$

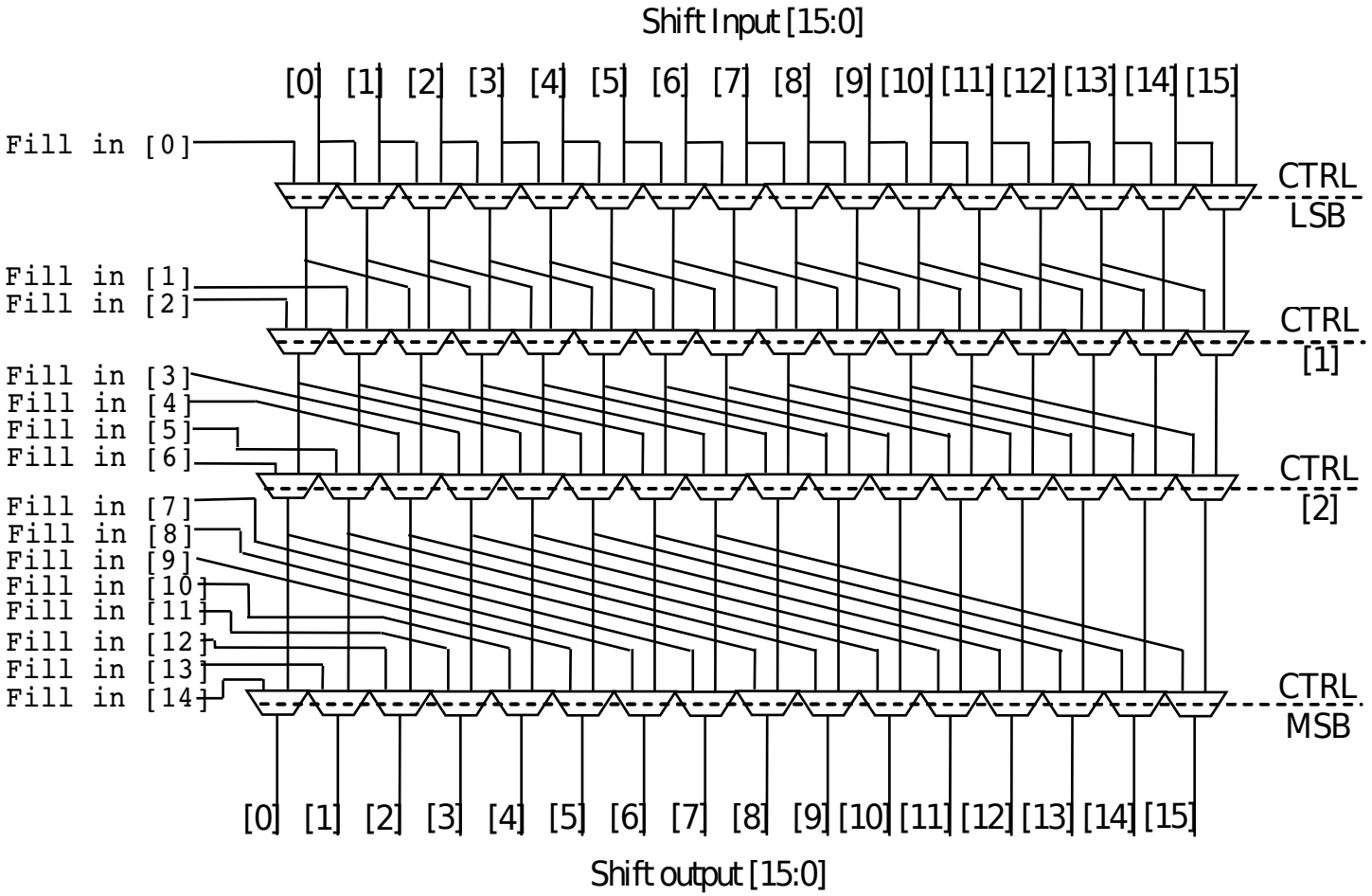
- How many operations do we need for $|A| - |B|$?
- The easy approach:
  - One adder for $|A|$
  - One adder for $|B|$
  - One adder for the final subtraction

Typical ALU shift operations



[Liu2008]

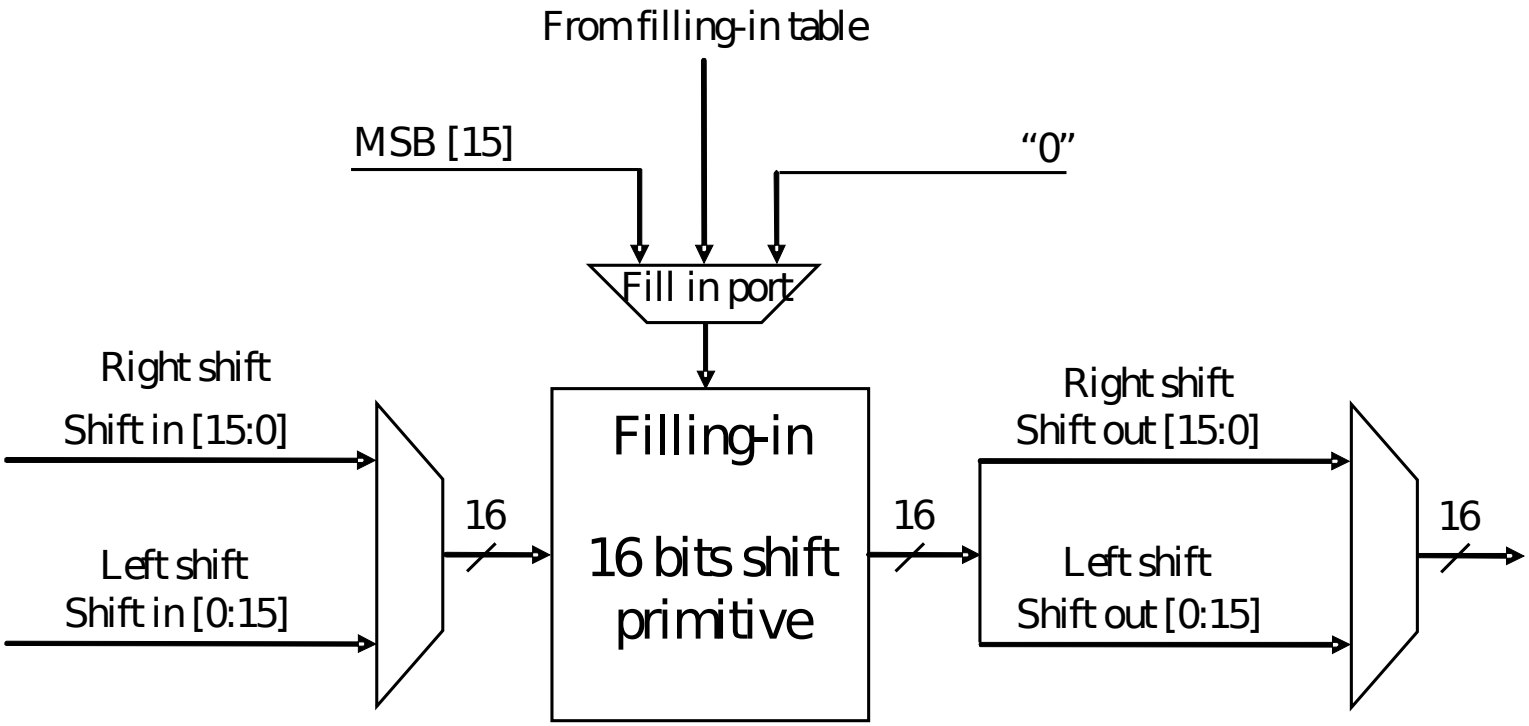
Shifter primitive



[Liu2008]

- Note: Barrel shifters based on 4-to-1 multiplexers may be more efficient

Hardware multiplexing in shifter



[Liu2008]

- Note: Fill in table may be complicated for some shift operations