# Computer Architecture and Organization

## Arithmetic
## Datapath & Control

## Lecture 4



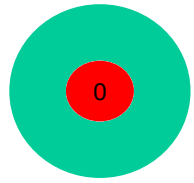


## Where are we?

- What did you learn last week?
  - You can design combinational logic (including Karnaugh map)
  - You can design synchronous sequential logic (including FSM)
  - You understand the different number representations
- This week:
  - Today:
    - You can perform simple arithmetic operations on unsigned and signed numbers and how it can be realized
    - You understand the components of a processor
  - Tuesday:
    - You can write programs in assembly for the ARC processor
  - Wednesday:
    - You understand the details of the data path and controller of the ARC processor

## Multiple choice question

- Try it yourself (do not discuss it with your neighbor)
- You have 10 seconds!
- What is de decimal value of pattern 1101
  - A: 13
  - B: -2
  - C: -3
  - D: -5
  - E: I don't know

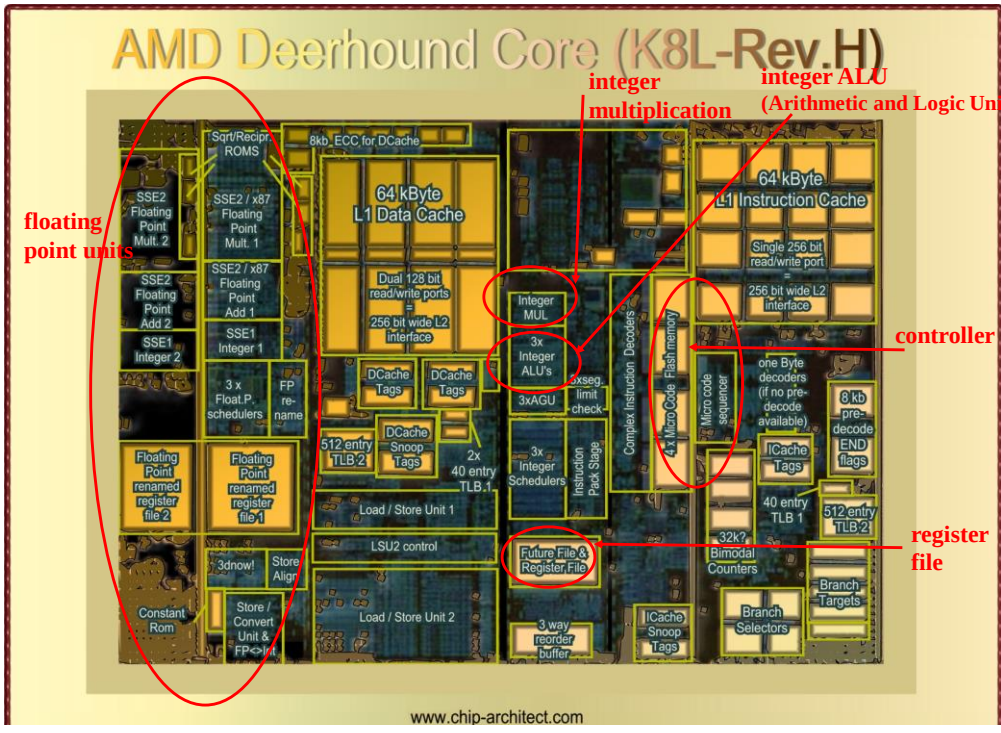







## Topics of this lecture

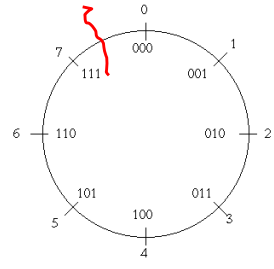
- Unsigned/Signed addition⇦
  - Ripple carry adder
  - Status signals
- Unsigned Multiplication
- Components of a processor
  - Datapath & Control
  - Register file
  - Shifter




5

# Addition of N-bit unsigned numbers

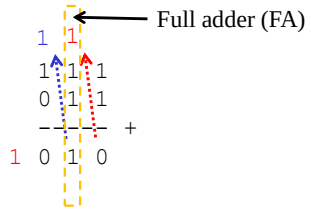
- Similar as addition in base 10 (primary school)
- Add two N-bit numbers and the result is N-bits.
- If the red line in the number circle is crossed the number can not be represented (overflow).

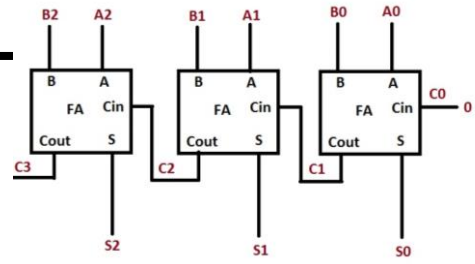



```
  1 1 1
  0 1 1
  ----- +
1 0 1 0
```

- When a carry-out is generated on MSB (most significant bit) → Overflow
- What if you want a N+1 bits result?
  - Extend both operands with 0 (although this is implicit)

# Ripple-carry adder



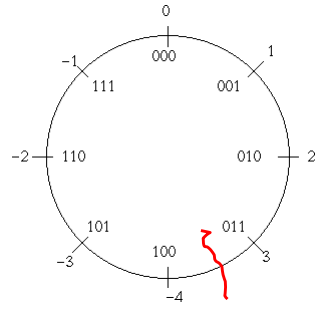




- A full adder (FA) has 3 inputs (a, b, cin) and 2 outputs (s, cout)
- You can design the logic of a full-adder! How?
  - Truth-table → Karnaugh-map for the 2 outputs
    → 2 level logic (delay ~2Δ)
- A chain of FA's is a ripple-carry adder.
  - What is the propagation delay of an N-bit ripple-carry adder?
    ~N × 2Δ
    (EE-students will reduce the delay by adding extra logic; week 5)





# Addition of N-bit signed numbers




- Add two N-bit numbers and the result is N-bits.
- If the red line in the number circle is crossed the number can not be represented (overflow).

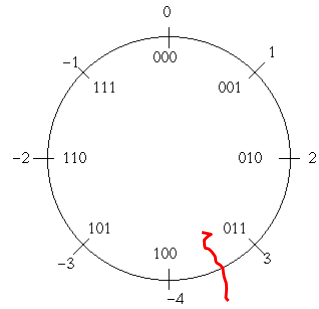
Bit position 1

```
0 1 1    (3)
1 1 0    (-2)
-----  +
x 0 1
```

- Bit position 1 generates a cout, therefore in bit position 2: $2^2$ (=cout) + $-2^2$ (=sign bit) = 0 → X is 0
- Also a FA can be used for MSB position?
  → $s_2$=0 and $cout_2$=1 and ignore this $cout_2$
  → result is correct
- Again a ripple-carry adder! Same hardware as for unsigned numbers.
- How detect overflow?

# Addition of N-bit signed numbers; cont'd

```
      0 1 1   (3)          1 1 1   (-1)
      0 1 0   (2)          1 0 0   (-4)
      ----- +              ----- +
(0)   1 0 1   (-3)   (1)   0 1 1    (3)
```




- Carry-ripple adder is used. Cout is ignored.
- Addition of a positive number and a negative number cannot result in an overflow.
- Overflow when (see both examples):
  - Two positive numbers are added and the result is negative, or
  - Two negative numbers are added and the result is positive
- You can derive logic for the overflow detection
  - Truth-table → Karnaugh-map



9

# Addition of 3-bit signed numbers: overflow

- Alternative detection of overflow using the carry bits
- Example: 3 bits numbers are sign extended (decimal value is not changed)

```
    1 1 1 1 1      carry
 .. 1 1 1 1 1 1 0   (-2)
 .. 1 1 1 1 1 1 1   (-1)
 ----------------- +
 .. 1 1 1 1 1 0 1   (-3)
No overflow (N=3)
```

```
    0 0 0 0 0      carry
 .. 0 0 0 0 0 0 0 1   (1)
 .. 0 0 0 0 0 0 1 0   (2)
 ----------------- +
 .. 0 0 0 0 0 0 1 1   (3)
No overflow (N=3)
```

```
    0 0 0 0 1      carry
 .. 0 0 0 0 0 0 1 0   (2)
 .. 0 0 0 0 0 0 1 1   (3)
 ----------------- +
 .. 0 0 0 0 1 0 0 1
Overflow (N=3)
```

```
    1 1 1 1 0      carry
 .. 1 1 1 1 1 1 1 0   (-2)
 .. 1 1 1 1 1 1 0 1   (-3)
 ----------------- +
 .. 1 1 1 1 0 0 1 1
Overflow (N=3)
```

- Overflow: Bit 2 of sum is not equal to bit 3 of sum (red)
  - Note: right most bit is Bit 0
- Overflow detection that is often used: cout1 ≠ cout2 (blue) (cout=carry out)
  - What is the logic for overflow detection?

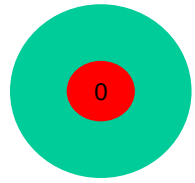


10

## Addition of numbers with different length

- Add these two signed numbers (20 sec)

```
      1 0 1
0 1 0 1 0
--------- +
```




- Sign extend the shortest operand

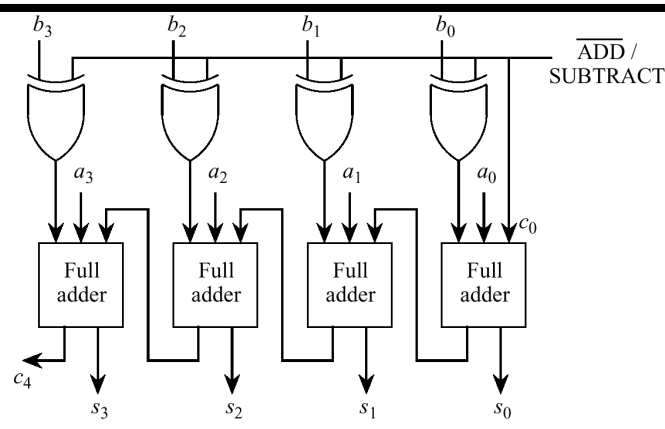
```
1 1 1 0 1    (-3)
0 1 0 1 0    (10)
--------- +
0 0 1 1 1     (7)
```



## Subtraction of signed numbers

- A-B is equal to A + (-B)
- How can you determine (-B)?
  - Invert all bits of B (=1-complement) and add 1 (2-complement of B)

## Ripple adder/subtractor




- This circuit can calculate A+B and A-B



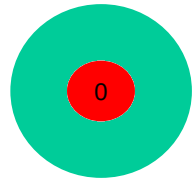


## Summary addition

- Unsigned numbers
  - Carry-ripple adder
  - Overflow when MSB generates carry out
  - If the numbers have different length → zero extend shortest vector.
- Signed numbers
  - Carry ripple adder
  - Overflow when carry generated on bit position N-2 is not equal to carry generated on bit position N-1.
  - If the numbers have different length → sign extend shortest vector.

# How does logic supports a program?

while (op1 < op2)
{ do something }

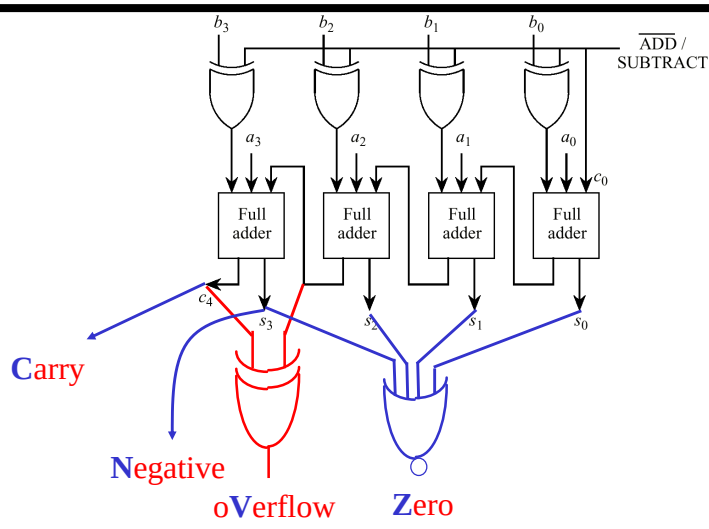



- **A program usually contains branches**
  - **How can a processor check the condition: op1 < op2 ?**
  - **op1 < op2 ≡ op1 – op2 < 0**
    **Perform the subtraction and check if the result is negative (status)! Based on the status the controller will instruct the data path.**
- **The status bits (sufficient for this course)**
  - **N (negative)**
  - **Z (zero)**
  - **V (oVerflow)**
  - **C (carry)**

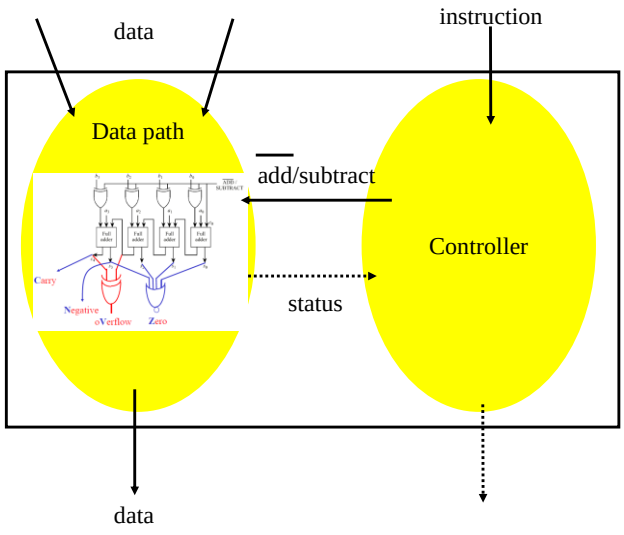




# Ripple carry add/subtract with status



# Incomplete simple processor




- Data path
  - Transport of data
  - Operation on data (add, and, shift,..)
  - Local storage of data (register file)
- Controller
  - Instructs the data path
  - Which operation?
  - and on which data?





# Topics of this lecture

- Unsigned/Signed addition
- Unsigned Multiplication ⇦
- Components processor
  - Datapath & Control
  - Register file
  - Shifter

# MULTIPLY (unsigned)

- Paper and pencil example (primary school ☺):

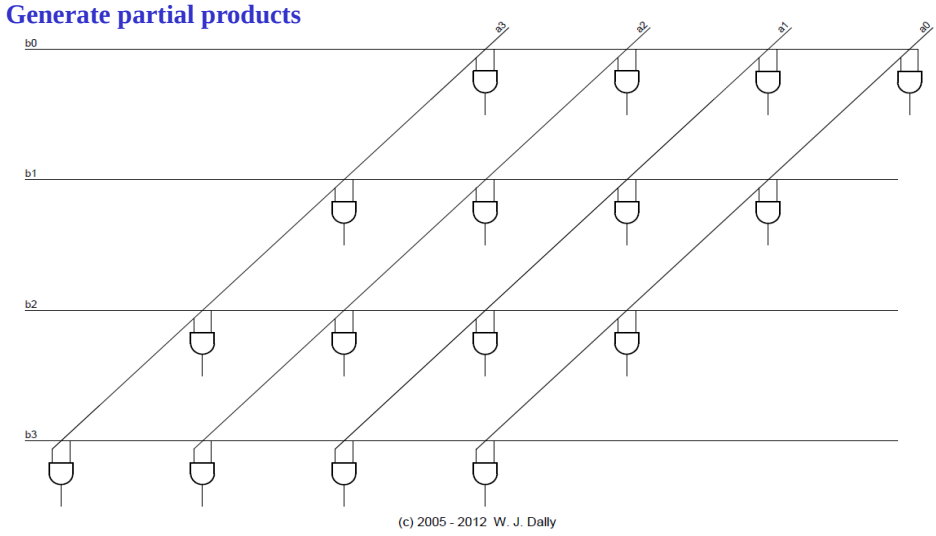
```
Multiplicand    1101
Multiplier      1011
                1101
               1101
              0000
             1101
Product     10001111
```

- m bits × n bits = m+n bit product (no overflow)
- Binary makes it easy:
  - 0 => place 0000..0 ( 0 × multiplicand)
  - 1 => place a copy multiplicand ( 1 × multiplicand)
- 2 versions of multiply hardware & algorithm:
  - combinational / sequential (shift-add algorithm)

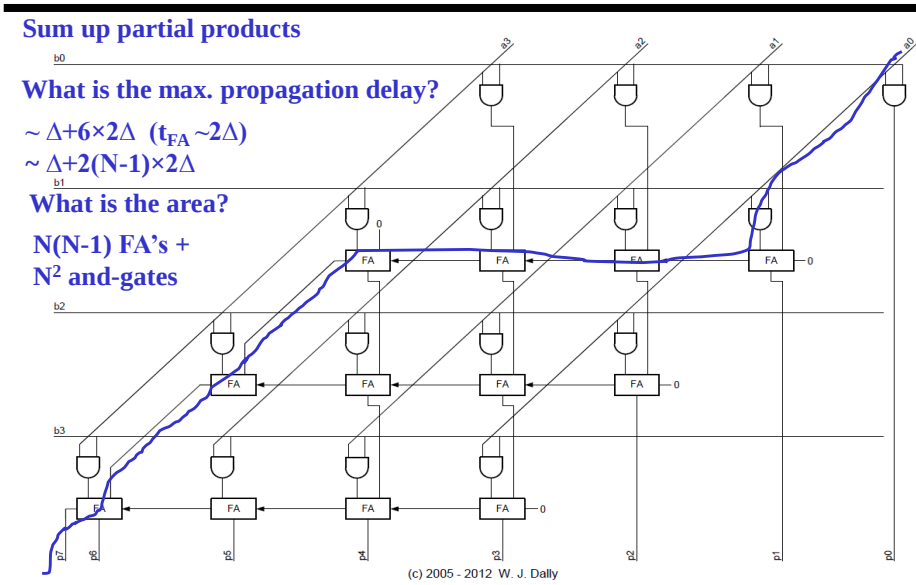




# Unsigned Combinational Multiplier



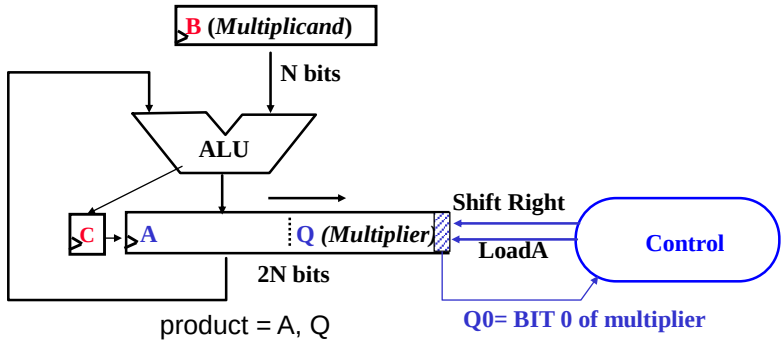
# Unsigned Combinational Multiplier, cont'd






# Sequential multiplier (shift-add method)

- Multiplicand N-bits, lower N bits of product register (2N bits) is initially filled with the multiplier.
- If Q0=1 (LSB) perform addition of multiplicand and higher N bits of product. Shift "C,A,Q" register one position to the right

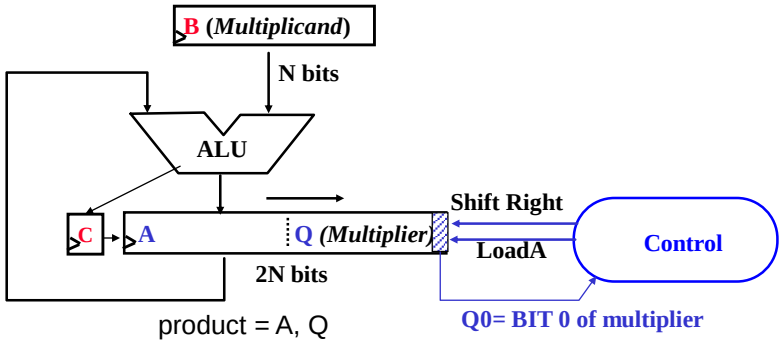





22

# Sequential multiplier (shift-add method)

- How many clock cycles?
  - 1 for initialization and 2N for the algorithm (assuming add admd shift are not in the same clock cycle)
- Area?
  - 1 N-bit Adder (in ALU), registers and control logic








# How multiply signed numbers?

- Size operands: m bits and n bits; result: m+n bit product (no overflow)
- For addition we can use the same hardware unsigned and signed numbers
- Can we use the same hardware for unsigned and signed multiplication?
- Why is this not correct?

```
      1 1 0
      1 0 1
      ----- x
      1 1 0
    0 0 0
  1 1 0
  ---------
? 1 1 1 1 0
```

# How multiply signed numbers? Cont'd

- Remember addition:
  signed numbers with different length?
  → extend with sign bits
- MSB of multiplier has weight $-2^2$.
  You have to add:
  → $-2^2 \times$ multiplicand
  → $2^2 \times$ (- multiplicand)
  → $2^2 \times$ (two's complement of multiplicand)

```
          1 1 0
          1 0 1
          ----- x
1 1 1 1 1 0
0 0 0 0 0 0
0 0 1 0 0 0
---------- +
0 0 0 1 1 0
```

- Easy isn't it?

Signed multiplication is not part of CAO.
Only remember that it is different from unsigned multiplication
EE students will use an alternative method in DH

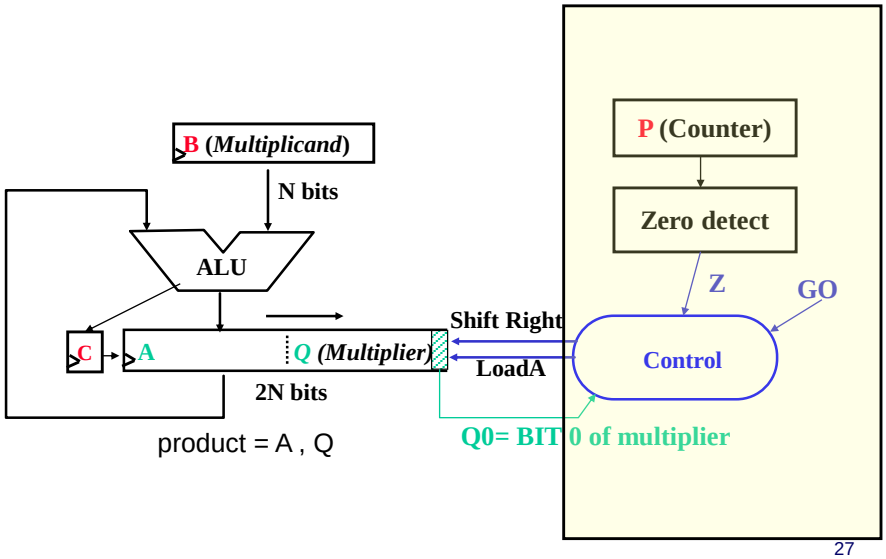




# Topics of this lecture

- **Unsigned/Signed addition**
  - **Ripple carry adder**
  - **Status signals**
- **Multiplication**
  - **Unsigned**
- **Components of a processor**
  - **Datapath & Control ⇦**
  - **Register file**
  - **Shifter**

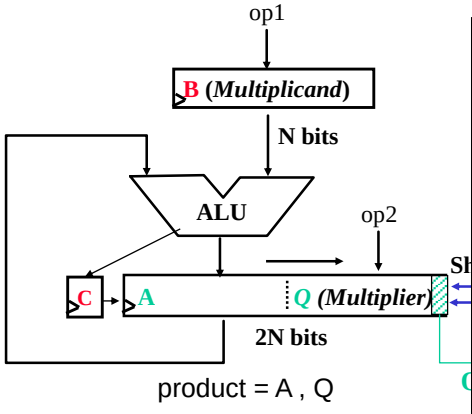
## Shift-add algorithm




27



## Datapath




The datapath performs the data processing.
Components of the datapath are:

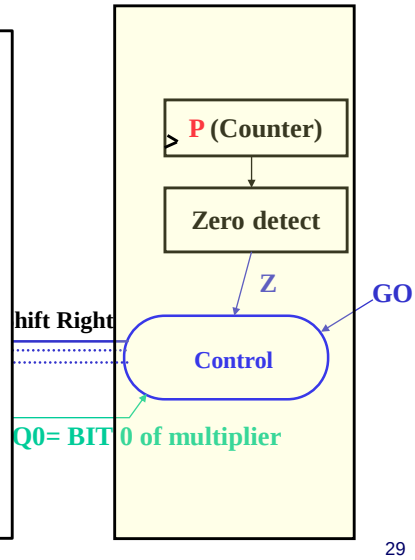
- ALU: Arithmetic and Logic unit
- Buses for transport of data
- Multiplexers for selection of data
- Registers for intermediate storage of data (e.g. register file).

28

# Controller

Controller:
- Determines what operation is performed in the datapath
- Receives status information from datapath

Two controller realizations:
- Hard wired
- Micro controlled

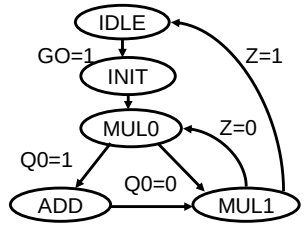






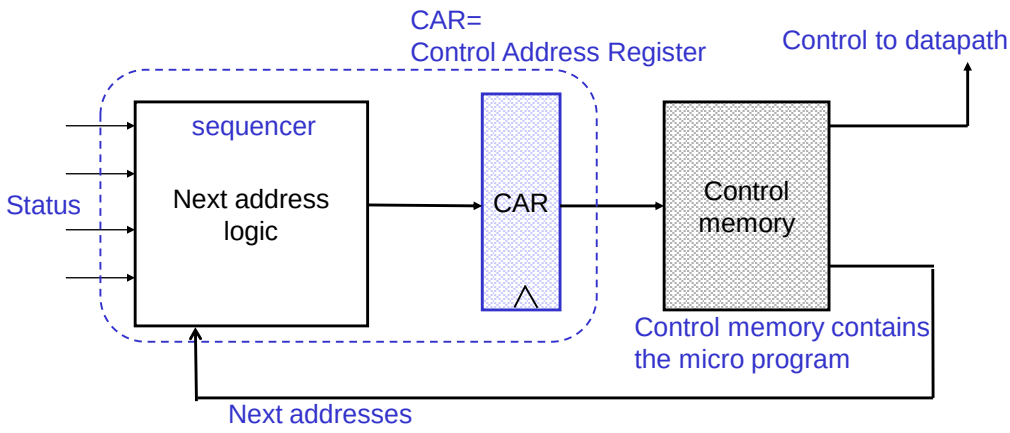

# State-machine controller for multiplier

- **State IDLE**
  - **No activity in datapath**
- **State INIT**
  - **Initialize registers**
- **State MUL0**
  - **No activity in datapath. Based on Q0 next state is determined**
- **State ADD**
  - **Perform addition in datapath**
- **State MUL1**
  - **Shift product register**




Note: it is assumed that addition and shifting cannot be done in the same clock cycle

## Micro programmed control unit

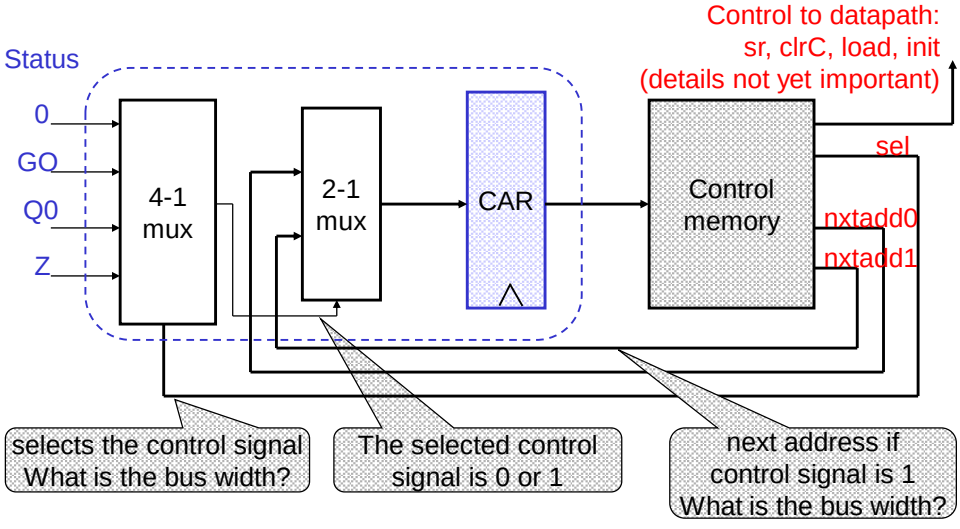



Is this a Mealy or Moore machine?





## Implementation for the multiplier



# Format microinstruction

| 11 – 9 | 8 – 6 | 5 – 4 | 3 – 0 |
|---|---|---|---|
| nxtadd1 | nxtadd0 | sel | Datapath control |

| Sel | name | operation |
|---|---|---|
| 00 | nxt: | CAR <= nxtadd0 |
| 01 | G: | IF GO=0 THEN CAR <= nxtadd0 ELSE CAR <= nxtadd1 |
| 10 | Q: | IF Q0=0 THEN CAR <= nxtadd0 ELSE CAR <= nxtadd1 |
| 11 | Z: | IF Z=0 THEN CAR <= nxtadd0 ELSE CAR <= nxtadd1 |

| Datapath control | |
|---|---|
| bit 3 | sr |
| bit 2 | clrC |
| bit 1 | load |
| bit 0 | init |

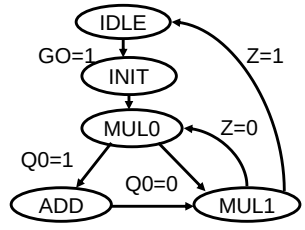


33

# Micro program for multiplier

Microprogram in bits ('red' bits are don't care)

| Address | nxtadd1 | nxtadd0 | SEL | datapath |
|---|---|---|---|---|
| 000 | 001 | 000 | 01 | 0000 |
| 001 | 000 | 010 | 00 | 0101 |
| 010 | 011 | 100 | 10 | 0000 |
| 011 | 000 | 100 | 00 | 0010 |
| 100 | 000 | 010 | 11 | 1100 |




Symbolic names for readability

| Address | nxtadd1 | nxtadd0 | SEL | datapath |
|---|---|---|---|---|
| idle | init | idle | Go | none |
| init | - | mul0 | nxt | init, clrC |
| mul0 | add | mul1 | Q0 | none |
| add | - | mul1 | nxt | load |
| mul1 | idle | mul0 | Z | sr_dec_cnt, clrC |



34

## Advantages of microprogramming

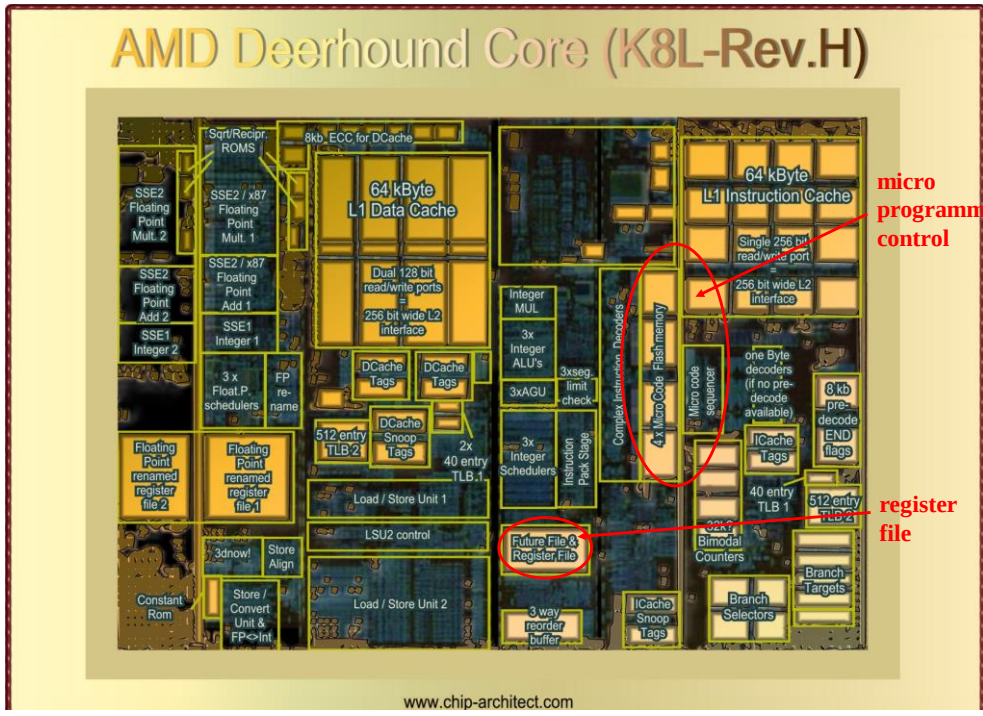
- Systematic
- Changes are made in the control memory
  - Upgrade of firmware ….
    Why shouldn't you interrupt the upgrade procedure of the firmware?
- Complex state machines can be implemented
- When control memory is RAM changes on-the-fly
- Restriction in this control unit:
  - Only 2 possible next states
  - Often 1 next address field is in the control memory and the other next address is the incremented value of CAR (e.g. ARC-processer).
    - Advantage: reduces the control memory size
    - Disadvantage: only one 'real' jump





## Topics of this lecture

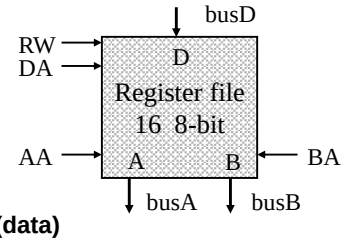
- **Unsigned/Signed addition**
  - **Ripple carry adder**
  - **Status signals**
- **Multiplication**
  - **Unsigned**
- **Components of a processor**
  - **Datapath & Control**
  - **Register file ⇦**
  - **Shifter**




37

# Storage Element: Register File; an example

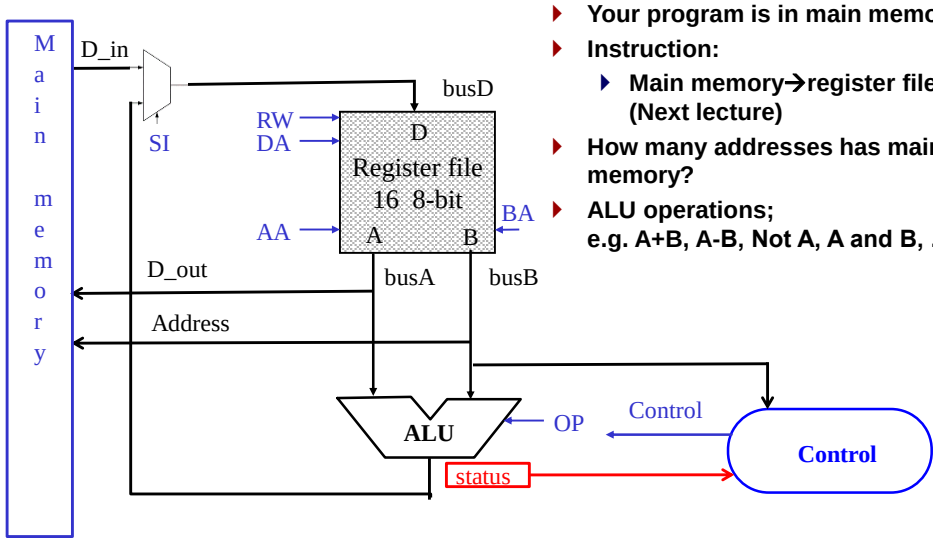
- A register file is often part of the Datapath for local storage of intermediate data.
- Register File has 16 registers:
  - Two 8-bit output busses: busA and busB
  - One 8-bit input bus: busD
- Register is selected by (all control signals!):
  - AA (address) selects the register to put on busA (data)
  - BA (address) selects the register to put on busB (data)
  - DA (address) selects the register to be written via busD when ReadWrite (RW) is 1
- ReadWrite (RW); a control signal!
  - The D input is used during a write operation (RW=1)
  - Read operation: the register file behaves as a combinational logic block
    - AA or BA valid => busA or busB valid after "access time"




38

38

# Simple processor

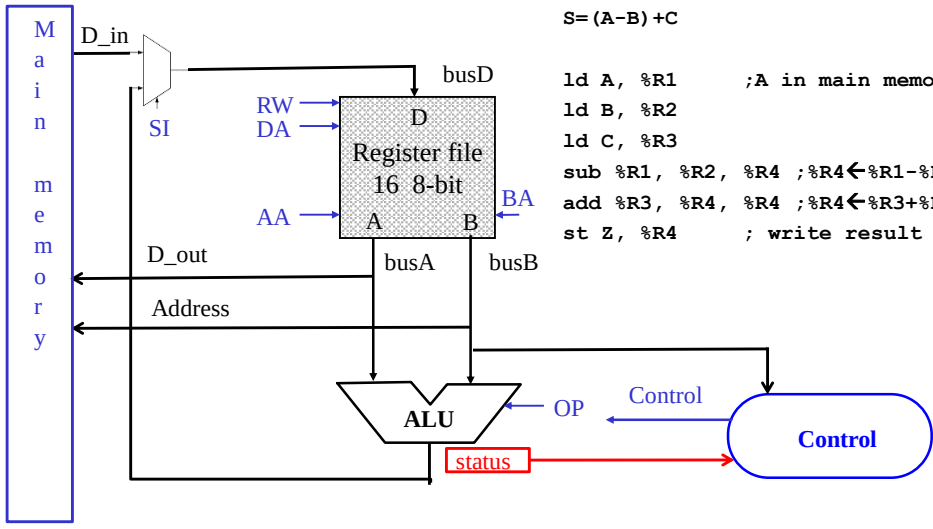



- Your program is in main memory
- Instruction:
  - Main memory→register file (Next lecture)
- How many addresses has main memory?
- ALU operations; e.g. A+B, A-B, Not A, A and B, …





# Our first program in assembly




```
S=A-B+C
S=(A-B)+C

ld A, %R1      ;A in main memory
ld B, %R2
ld C, %R3
sub %R1, %R2, %R4 ;%R4←%R1-%R2
add %R3, %R4, %R4 ;%R4←%R3+%R4
st Z, %R4      ; write result in Z
```

## Topics of this lecture

- **Unsigned/Signed addition**
  - **Ripple carry adder**
  - **Status signals**
- **Multiplication**
  - **Unsigned**
- **Components of a processor**
  - **Datapath & Control**
  - **Register file**
  - **Shifter ⇦**



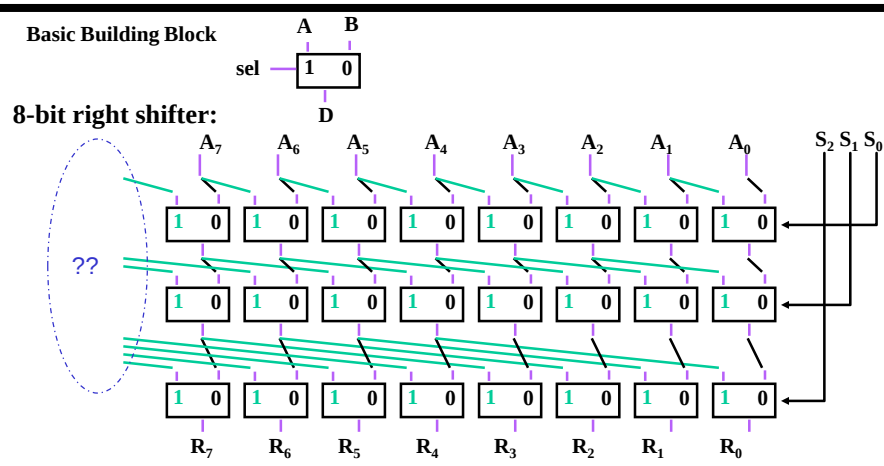
41

## Example of a floating point addition

$$-1.1100 \times 2^{-3} + 1.0000 \times 2^{-1}$$

- Step 1: alignment of fraction (exponent the same)
  $-1.1100 \times 2^{-3} = -0.0111 \times 2^{-1}$ (shift operand)
- Step 2: add fractions
  $-0.0111 + 1.0000 = 0.1001$
- Step 3: post normalization of result
  $0.1001 \times 2^{-1} = 1.0010 \times 2^{-2}$

# Barrel Shifter from MUXes




- Combinational or sequential logic?
- What comes in the MSBs?
- How many levels for 32-bit shifter?





# Topics of this lecture

- Unsigned/Signed addition
  - Ripple carry adder
  - Status signals
- Multiplication
  - Unsigned
- Components of a processor
  - Datapath & Control
  - Register file
  - Shifter

## Next lecture

- **Instruction Set Architecture (chapter 4)**
- **Download the ARCTools (see Canvas)**



46