ARM Cortex M3
Instruction Set Architecture

Gary J. Minden
March 29, 2016

Information and Telecommunication Technology Center

KU THE UNIVERSITY OF KANSAS

Calculator Exercise

- Calculate:

```
X = (45 * 32 + 7) / (65 – 2 * 18)
```

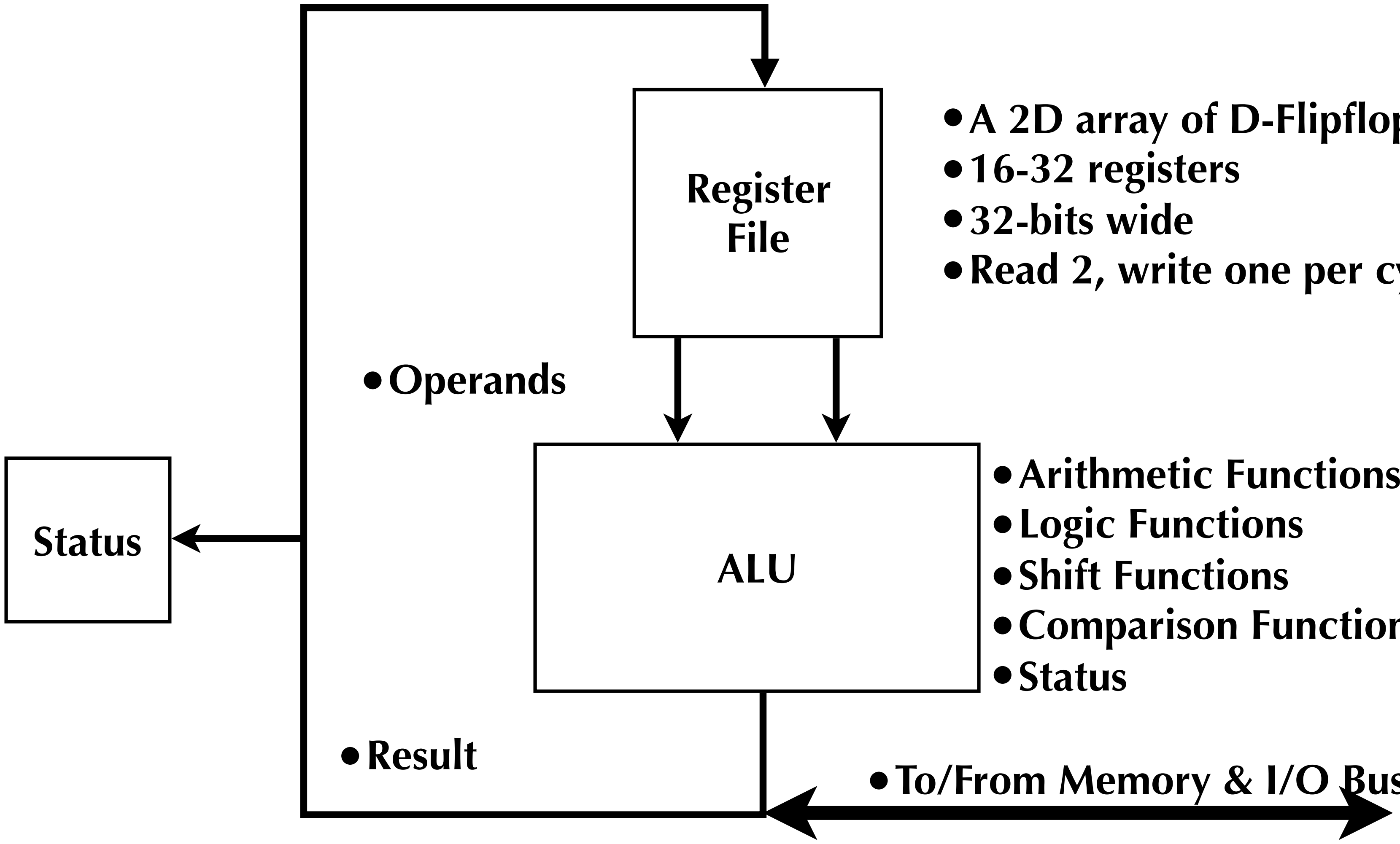
Instruction Set Architecture (ISA)

- ISAs define the instructions the hardware execute
  - Data types
  - Moving Data
  - Operations
  - Conditionals
  - Runtime structure, e.g. subroutines, interrupts, and system calls
- Review simple Load/Store instructions
- Addressing modes
- See: Texas Instruments, Cortex-M3 Instruction Set, TECHNICAL USER'S MANUAL at: http://www.ittc.ku.edu/~gminden/Embedded_Systems/PDFs/TI_ARM_AssemblerManual.pdf

Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

CPU Data Path

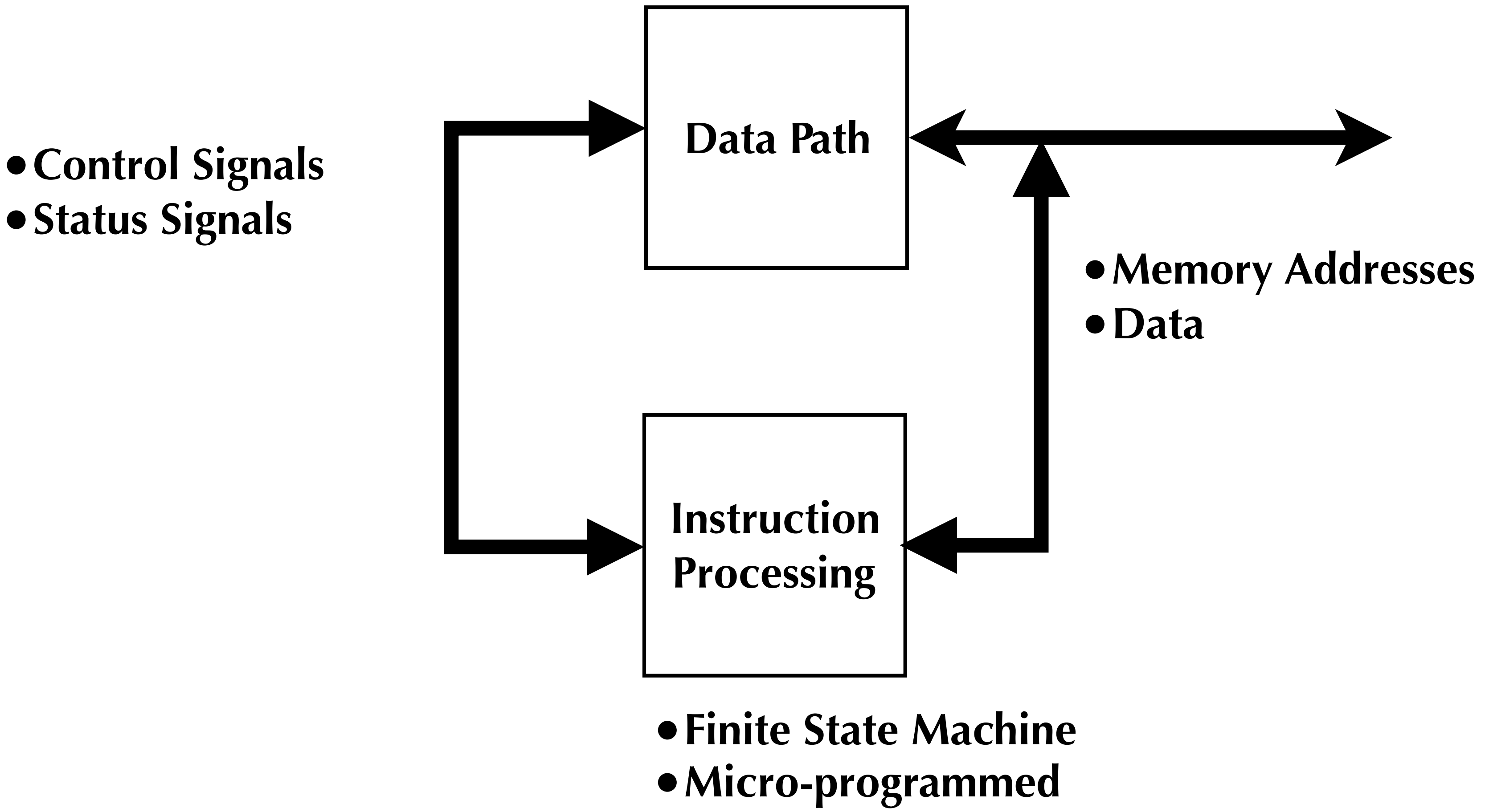


- A 2D array of D-Flipflops
- 16-32 registers
- 32-bits wide
- Read 2, write one per cycle

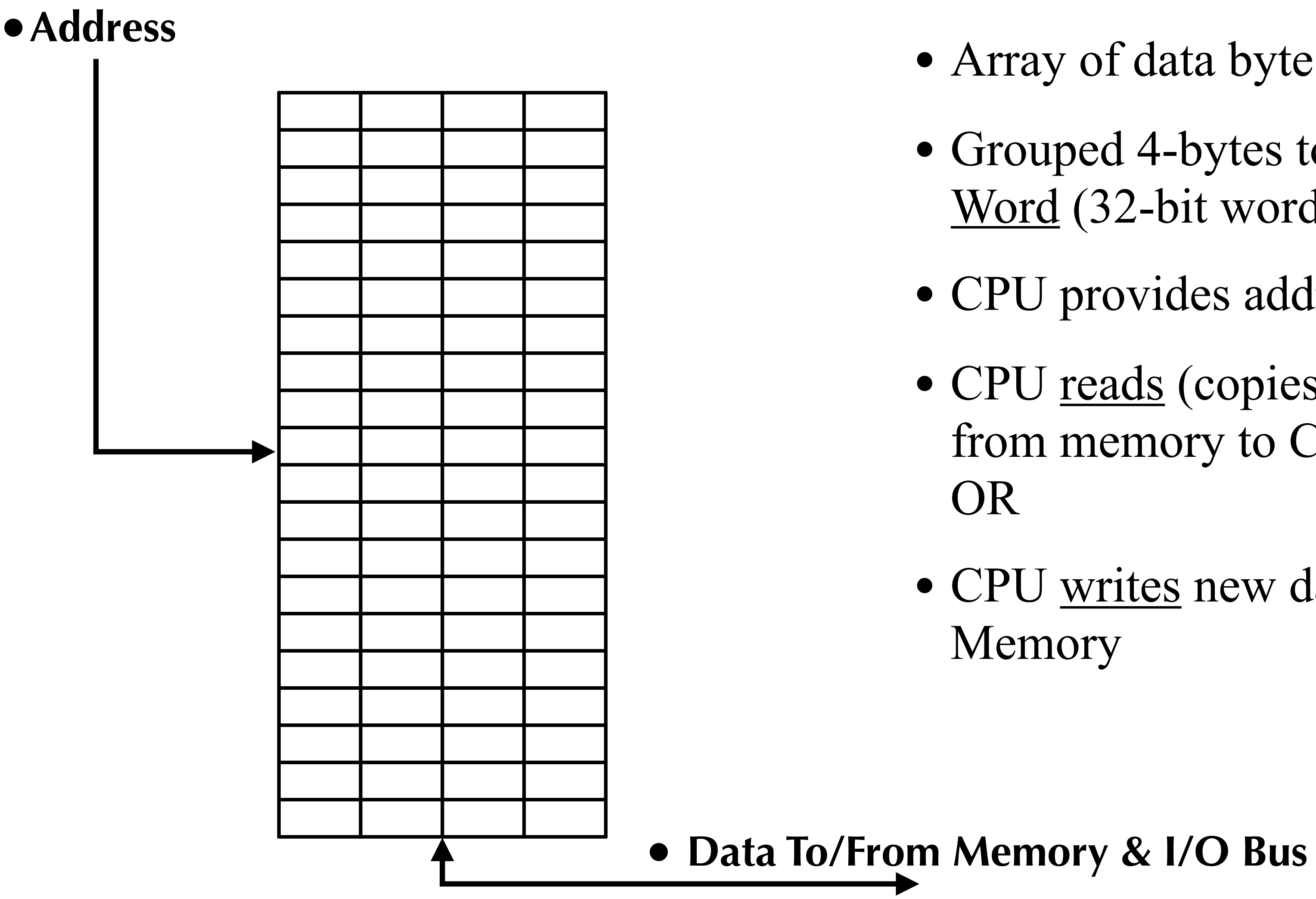
- Arithmetic Functions
- Logic Functions
- Shift Functions
- Comparison Functions
- Status

- To/From Memory & I/O Bus

CPU Controller



Memory Structure



- Array of data bytes
- Grouped 4-bytes to a Word (32-bit words)
- CPU provides address
- CPU reads (copies) data from memory to CPU OR
- CPU writes new data to Memory

Data Types

- Word (W) -- 32 bits
- Half Word (H) -- 16 bits
- Signed Half Word (SH) -- 16 bits extended to 32 bits
- Byte (B) -- 8 bits
- Signed Byte (SB) -- 8 bits extended to 32 bits
- Double (D) -- 64 bits

Information and Telecommunication Technology Center

KU THE UNIVERSITY OF KANSAS

Syntax Notation

- Angle brackets, <> Enclose alternative forms of the operand
- Braces, {} Enclose optional operands
- Op2 Is a flexible second operand that can be either a register or a constant

Loading (Copying) Data from Memory

- LDR loads (copies) data from memory to a register
- LDR{type}{cond}   Rt,Rn
  - Rt is destination register
  - Rn is source operand
- Common for a register to hold memory address
  LDR{type}{cond}   Rt,[Rn]

```
LDR       R5,[R9]    ; Load word
LDRB      R5,[R9]    ; Load Byte
LDRSB     R5,[R9]    ; Load sign
                     ;  extended Byte

IT        EQ         ; Cond. setup
LDREQ     R5,[R9]    ; Cond. execution
```

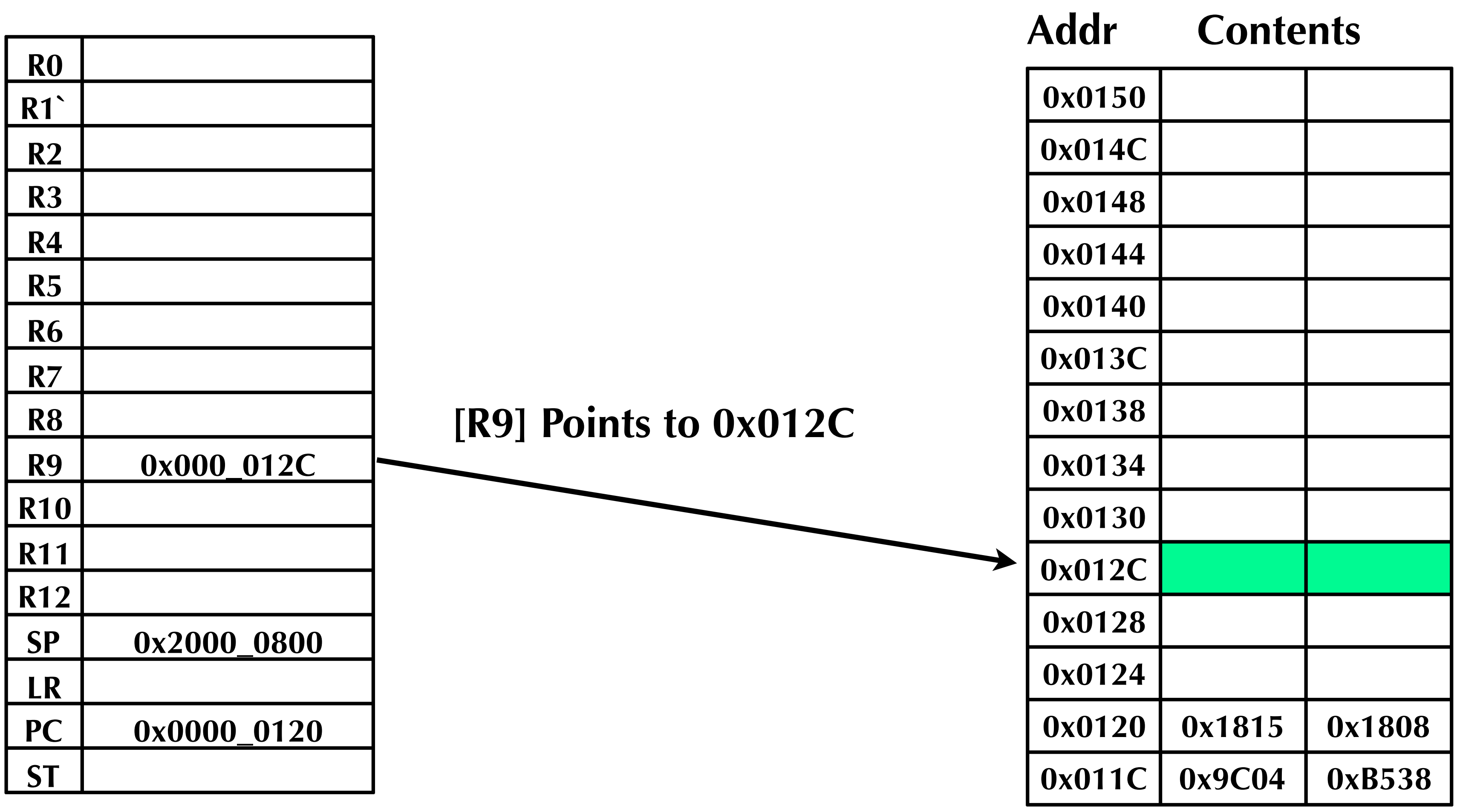
Kinds of Op2

- Register holds memory address
  - [Rn]
- Offset addressing
  - The offset value is added to address obtained from the register Rn. The result is used as the address for the memory access. Offset must be in the range -255 to +4095. The register Rn is not changed. The assembly language syntax for this mode is:
  - [Rn, #offset]

Pre- and Post-Indexing

- Pre-indexed addressing
  - The offset value is added to the address obtained from the register Rn. The result is used as the address for the memory access **and written back into the register Rn**. Offset must be in the range -255 to +4095. The assembly language syntax for this mode is:
  - [Rn, #offset]!
  - Increment address first, update Rn, then use as memory address
- Post-indexed addressing
  - The the contents of Rn are used as the memory address. Then the offset value is added to the register Rn. **Rn is updated after use**. Offset must be in the range -255 to +4095. The assembly language syntax for this mode is:
  - [Rn], #offset
  - Use Rn as memory address, add offset, update Rn

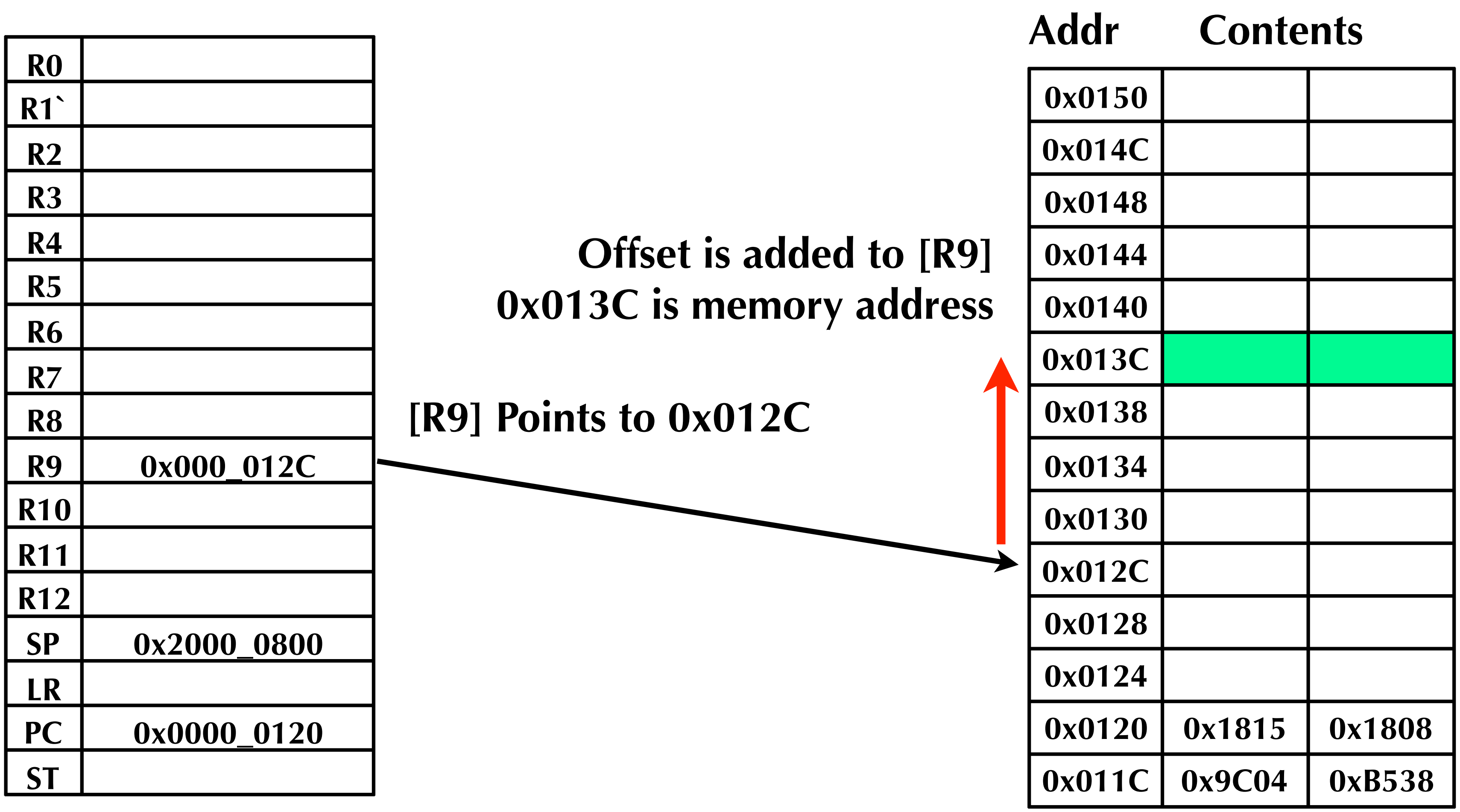
LDR Rt,[Rn] -- Immediate Offset



R0	
R1`	
R2	
R3	
R4	
R5	
R6	
R7	
R8	
R9	0x000_012C
R10	
R11	
R12	
SP	0x2000_0800
LR	
PC	0x0000_0120
ST	

Addr	Contents	
0x0150		
0x014C		
0x0148		
0x0144		
0x0140		
0x013C		
0x0138		
0x0134		
0x0130		
0x012C	🟩	🟩
0x0128		
0x0124		
0x0120	0x1815	0x1808
0x011C	0x9C04	0xB538

LDR Rt,[Rn,#16] -- Immediate Offset



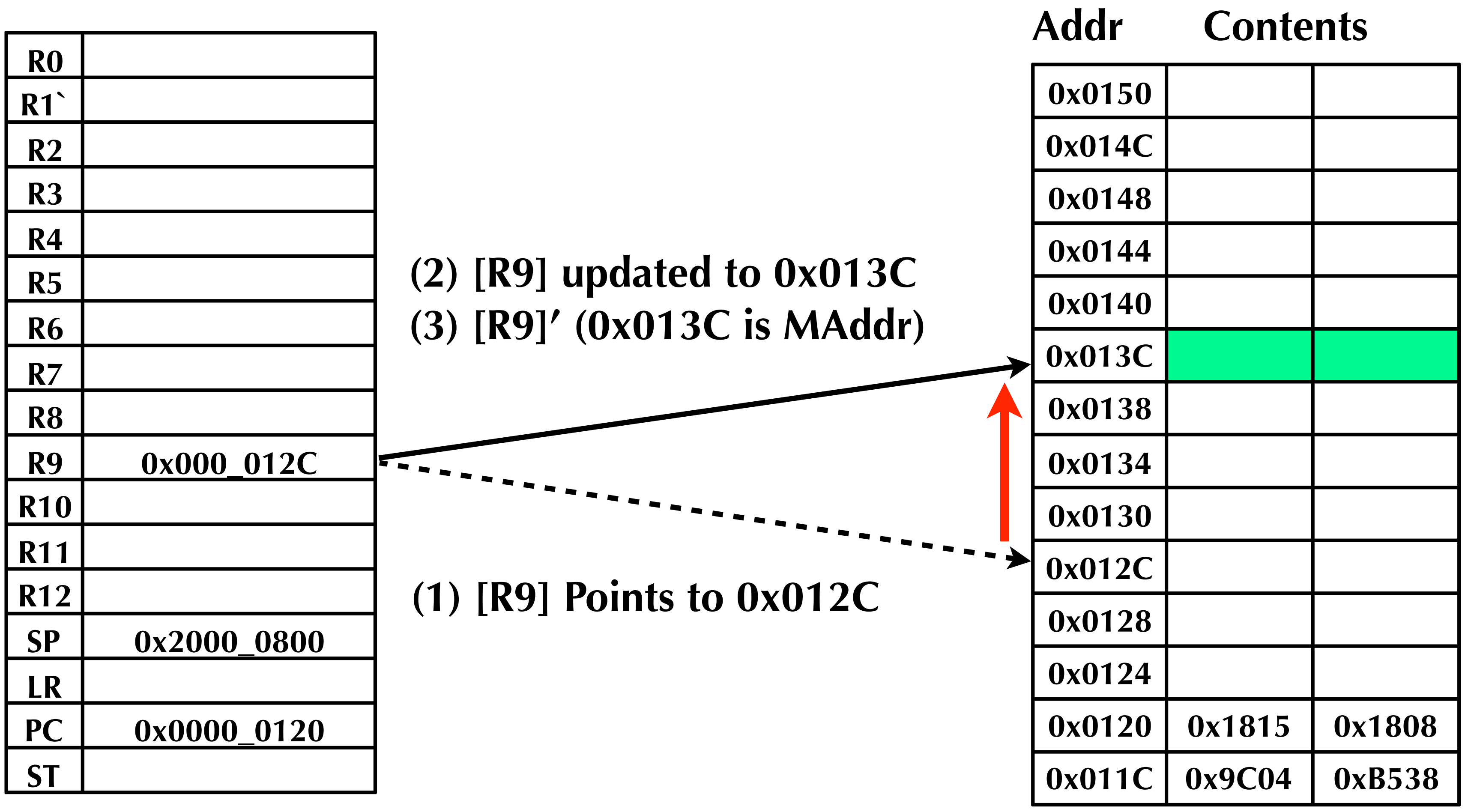
R0	
R1`	
R2	
R3	
R4	
R5	
R6	
R7	
R8	
R9	0x000_012C
R10	
R11	
R12	
SP	0x2000_0800
LR	
PC	0x0000_0120
ST	

Offset is added to [R9]
0x013C is memory address

[R9] Points to 0x012C

Addr	Contents	
0x0150		
0x014C		
0x0148		
0x0144		
0x0140		
0x013C	🟩	🟩
0x0138		
0x0134		
0x0130		
0x012C		
0x0128		
0x0124		
0x0120	0x1815	0x1808
0x011C	0x9C04	0xB538

LDR Rt,[Rn,#16]! -- Pre-Indexed



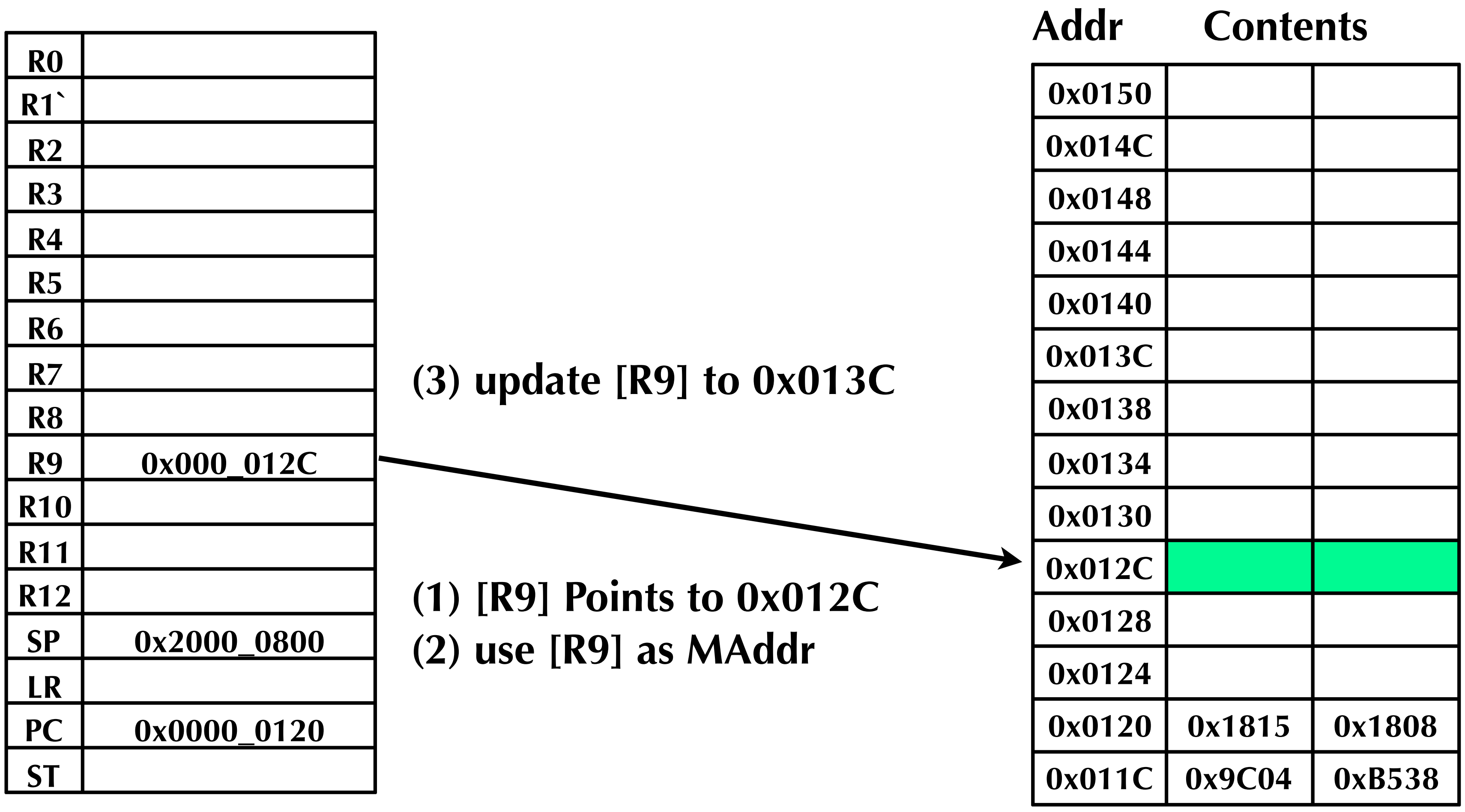
R0	
R1`	
R2	
R3	
R4	
R5	
R6	
R7	
R8	
R9	0x000_012C
R10	
R11	
R12	
SP	0x2000_0800
LR	
PC	0x0000_0120
ST	

(2) [R9] updated to 0x013C
(3) [R9]′ (0x013C is MAddr)

(1) [R9] Points to 0x012C

Addr	Contents	
0x0150		
0x014C		
0x0148		
0x0144		
0x0140		
0x013C	🟩	🟩
0x0138		
0x0134		
0x0130		
0x012C		
0x0128		
0x0124		
0x0120	0x1815	0x1808
0x011C	0x9C04	0xB538

LDR Rt,[Rn],#16 -- Post-Index



R0	
R1`	
R2	
R3	
R4	
R5	
R6	
R7	
R8	
R9	0x000_012C
R10	
R11	
R12	
SP	0x2000_0800
LR	
PC	0x0000_0120
ST	

(3) update [R9] to 0x013C

(1) [R9] Points to 0x012C
(2) use [R9] as MAddr

Addr	Contents	
0x0150		
0x014C		
0x0148		
0x0144		
0x0140		
0x013C		
0x0138		
0x0134		
0x0130		
0x012C	🟩	🟩
0x0128		
0x0124		
0x0120	0x1815	0x1808
0x011C	0x9C04	0xB538

LDR -- Register Offset

- A second source register, Op2, holds the offset
- Op2 can be a register, e.g. R8
- Op2 can be shifted left 0 to 3 bits

```
LDR       R5,[R9,R8]        ; MAddr <= R9 + R8
LDR       R5,[R9,R8,LSL #2] ; MAddr <= R9 + (4 * R8)
```

LDR -- PC Relative

- The PC is used as the base address
  - Offset is in the range +/- 4095

```
LDR         R5,Data1
LDR         R5,Table5
```

Store (Write) Register to Memory

- The STR instruction is similar to the LDR instruction
- Data is moved (copied) from the register file to memory

Computing an Address

- ADR loads a register with a memory address
- The address is the **PC+4**+Offset
- The offset is in the range +/-4095

```
ADR       R5,Data1
ADR       R5,Table5
```

PUSH and POP

- PUSH -- Move (copy) registers from the register file to the Stack
  - Highest register is at highest memory location
- POP -- Move (copy) memory data to the register file
- Used to save values
- Used for temporary storage, e.g. when you need more temporary values than you have working registers

General Data Processing Instructions

Addition and Subtraction

- ADD, ADC, SUB, SUBC, RSB
  - ADC and SUBC -- use the carry bit for multiple precision
  - RSB -- reverse subtract
  - op{type}{cond}   {Rd,} Rn, Operand2
  - op{type}{cond}   {Rd,} Rn, #imm12

```
ADD     R2,R1,R3
SUBS    R8,R6,#240   ; Set status flags
RSB     R4,R4,#1280 ; 1280-R4

ADDS    R4,R0,R2     ; Add least significant 32-bits
ADC     R5,R1,R3     ; Add most significant 32-bits plus carry
```

Bit-wise Operations

- AND, ORR, EOR, BIC, ORN
  - BIC -- equivalent to (AND NOT), clear bits in Rn which are set in Operand2
  - ORN -- equivalent to (OR NOT), set bits in Rn which are clear in Operand2
  - op{type}{cond}   {Rd,} Rn, Operand2

```
AND     R9,R2,#0xFF00    ; Keep byte 1 of R2
ANDS    R9,R8,#0x19      ; Set status bits
BIC     R0,R1,#0xAB      ; Clear specified bits of R1
```

Information and Telecommunication Technology Center

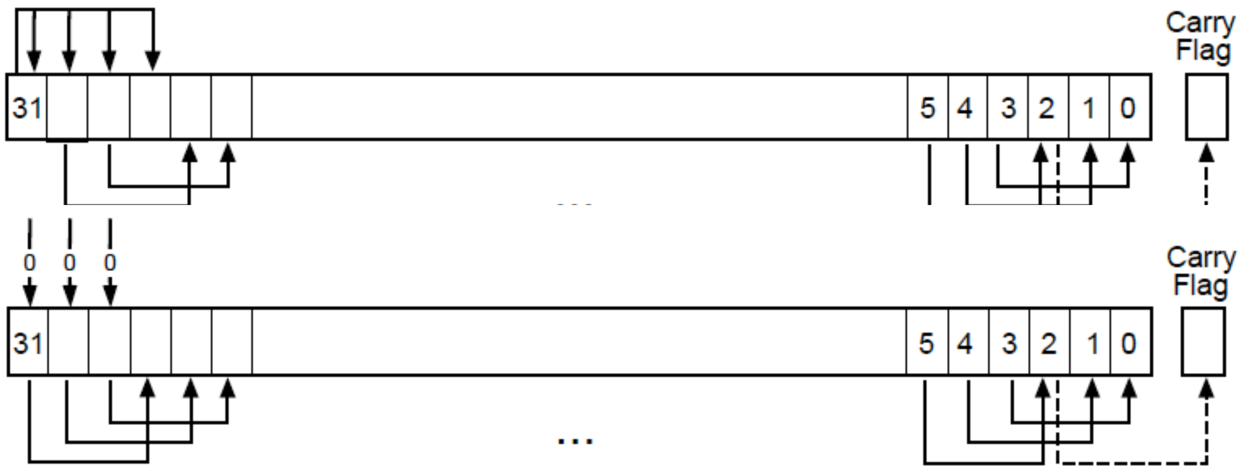
KU THE UNIVERSITY OF KANSAS

Shifts

- ASR, LSL, LSR, ROR, and RRX
  - ASR -- Arithmetic shift right
  - RRX -- move bits in register to the right by one position
  - op{type}{cond}   {Rd,} Rn, Rs
  - op{type}{cond}   {Rd,} Rn, #n
  - RRX   Rd, Rm

ASR

LSR



Count Leading Zeros

- CLZ
  - Count the number of leading zeros
  - op{cond} Rd, Rn

Compare and Test

- CMP, CMN, TST, TEQ
  - CMP -- Compute Rn - Operand2, set N, Z, C, V, discard result
  - CMN -- Compute Rn + Operand2, set N, Z, C, V, discard result
  - TST -- Compute Rn AND Operand2, set N and Z, discard result
  - TEQ -- Compute Rn XOR Operand2, set N and Z, discard result

```
CMP     R9,R2               ; Compare the value of R9 to R2

TST     R9,#x0003           ; Test if the bits 1..0 are 1
TEQ     R9,R7               ; Test if R9 is the same as R7
```

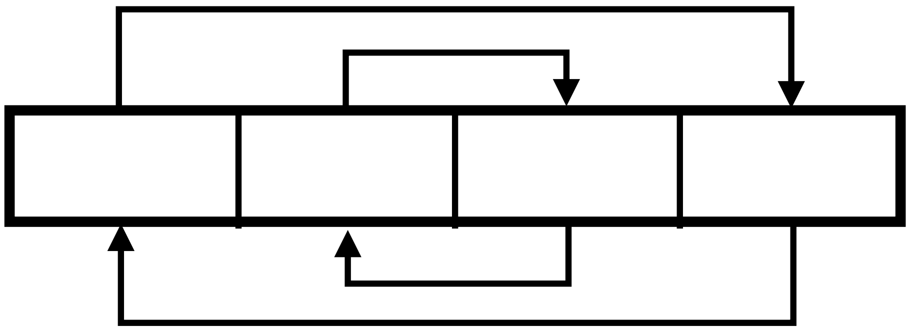
Move data

- MOV, MVN, MOVT
  - MOV -- copy the value of Operand2 to Rd
  - MVN -- copy the complement (NOT) of Operand2 to Rd
  - MOVT -- copy the value of Operand to the most significant (top) 16 bits of Rd
  - op{cond} Rd, Operand2
  - op{cond} Rd, #imm16

```
MOV    R9,R2          ; Copy R2 to R9
MVN    R9,R4          ; Copy the complement of R4 to R9
MOV    R9,#0x3400     ; Load R9 with a constant
MOVT   R9,#0xF0F0     ; Load the top half of R9 with #0xF0F0
```

Reverse Bytes and Bits

- REV, REV16, REVSH, RBIT
    - REV -- reverse byte order in a word
    - REV16 -- reverse byte order in each half word
    - REVSH -- reverse byte order in lower half word and sign extend
    - RBIT -- Reverse the bit order in the word
    - op{cond} Rd, Rn



Multiply and Multiply and Accumulate

- MUL, MLA, MLS
  - MUL -- computes Rn * Rm and puts least significant 32-bits of the product in Rd
  - MLA -- computes Rn * Rm, adds the value from Ra, and puts least significant 32-bits in Rd
  - MLS -- computes Rn * Rm, subtracts the value from Ra, and puts least significant 32-bits in Rd
  - op{cond} {Rd,} Rn, Rm
  - op{cond} {Rd,} Rn, Rm, Ra
  - MLA and MLS are used for fast implementation of digital signal processing functions, among other uses
  - UMULL, UMLAL, SMULL, SMLAL -- Signed and unsigned multiple producing 64-bit results

Divide

- SDIV and UDIV
  - SDIV, UDIV -- computes Rn / Rm and puts quotient in Rd, rounded towards 0
  - op{cond} {Rd,} Rn, Rm

Miscellaneous Instructions

- BFC and BFI -- clear a range of bits in a word
- SBFX and UBFX -- extract a range of bits from a word and possibily sign extend (SBFX)
- SXT and UXT -- sign extend a byte or half word to 32-bits

Branch Instructions

Branch Instructions

- B, BL, BLX, BX, CBZ, CBNZ
  - B -- branch (change PC) to destination address
  - BL -- copy PC to LR, branch (change PC) to destination address
  - BX -- branch to address in Rm
  - BLX -- copy PC to LR, branch to address in Rm
  - CBZ, CBNZ -- compare register to 0 and branch if zero (non-zero
  - op{cond}   label
  - op{cond}   Rm
  - op   Rn, label

```
B       EndIf               ; Branch to end of IF statement
BL      Sub_1               ; Branch and link to subroutine
BLX     R1                  ; Branch and link to subroutine,
                            ;   address is in R1
```

IF-THEN

- IT
  - Define tests for conditional execution
  - IT{x{y{z}}}   cond
  - “cond” for first instruction in block
  - x, y, z for second, third, and fourth instructions in block, respectively

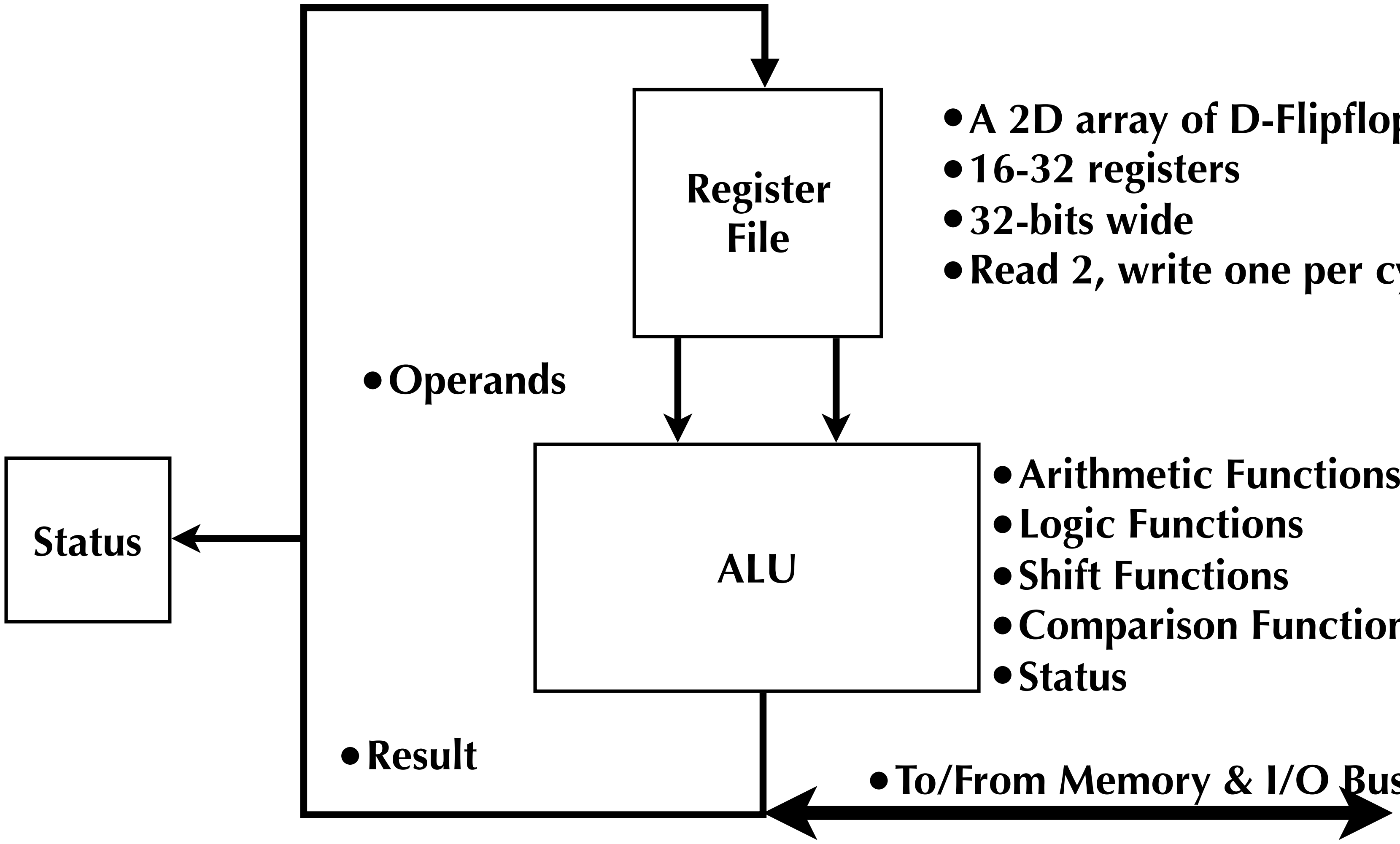
```
IT      GT            ; First condition is GT
BLGT    Sub_1         ; Branch and link to subroutine if GT

CMP     R0,#9         ; Compare R0 to 9
ITE     GT            ; Define conditions
ADDGT   R1,R0,#55     ; Convert 0xA -> ‘A’
ADDLE   R1,R0,#48     ; Convert 0x0 -> ‘0’
```

Miscellaneous

- BKPT -- Breakpoint, go to debug exception handler
- CPS -- Change processor state
- DMB -- Insure all memory accesses are completed
- DSB -- Insure all memory accesses are completed and hold the instruction pipeline
- ISB -- flush pipeline of processor, i.e. complete all previous instructions
- MSR and MRS -- move from/to special register
- NOP -- do nothing
- SEV and WFE -- send event to/wait for event from multiple processors
- SVC -- Supervisor call
- WFI -- Wait for interrupt

Programming

CPU Data Path



- A 2D array of D-Flipflops
- 16-32 registers
- 32-bits wide
- Read 2, write one per cycle

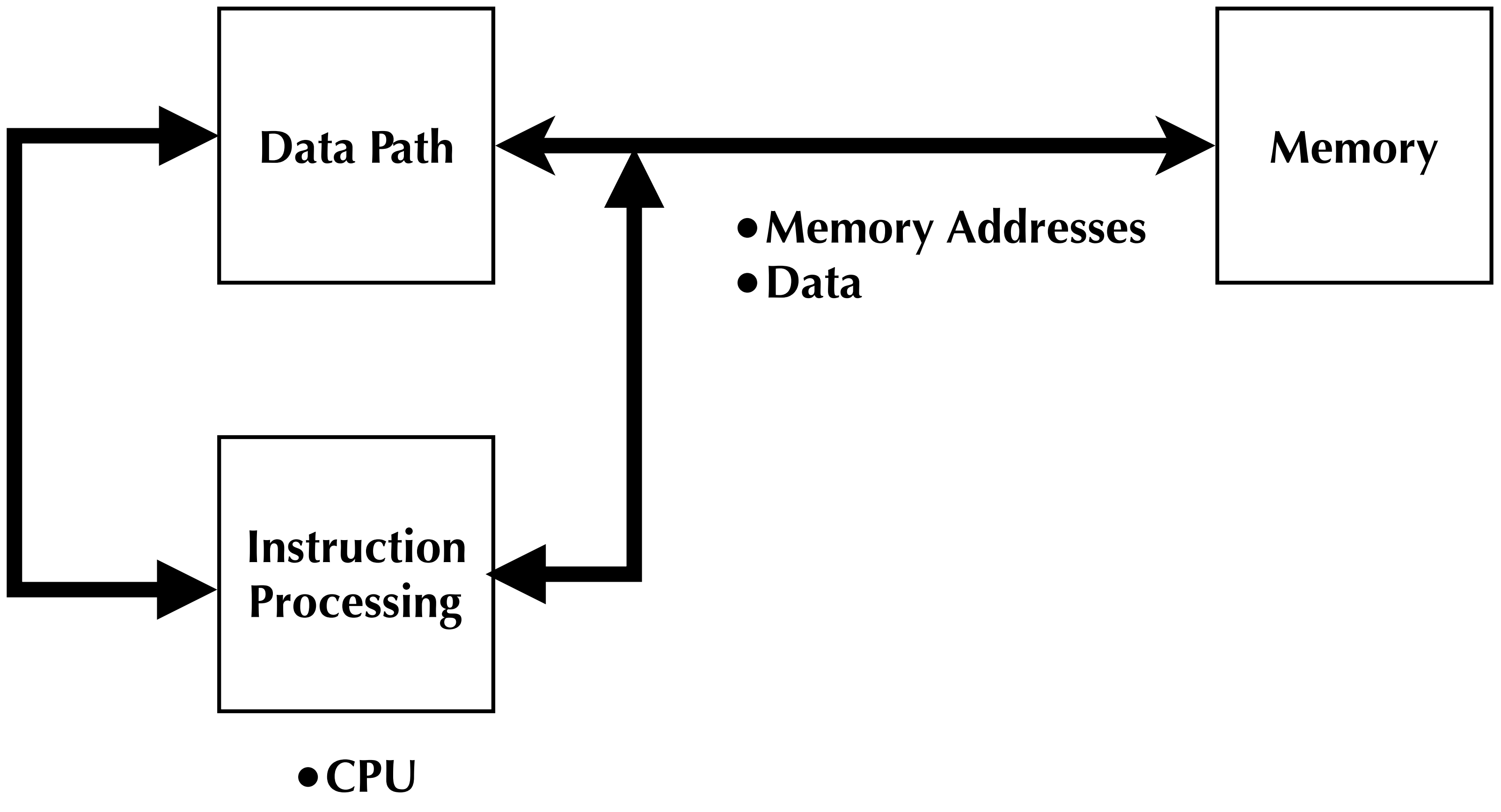
- Operands

- Arithmetic Functions
- Logic Functions
- Shift Functions
- Comparison Functions
- Status

- Result

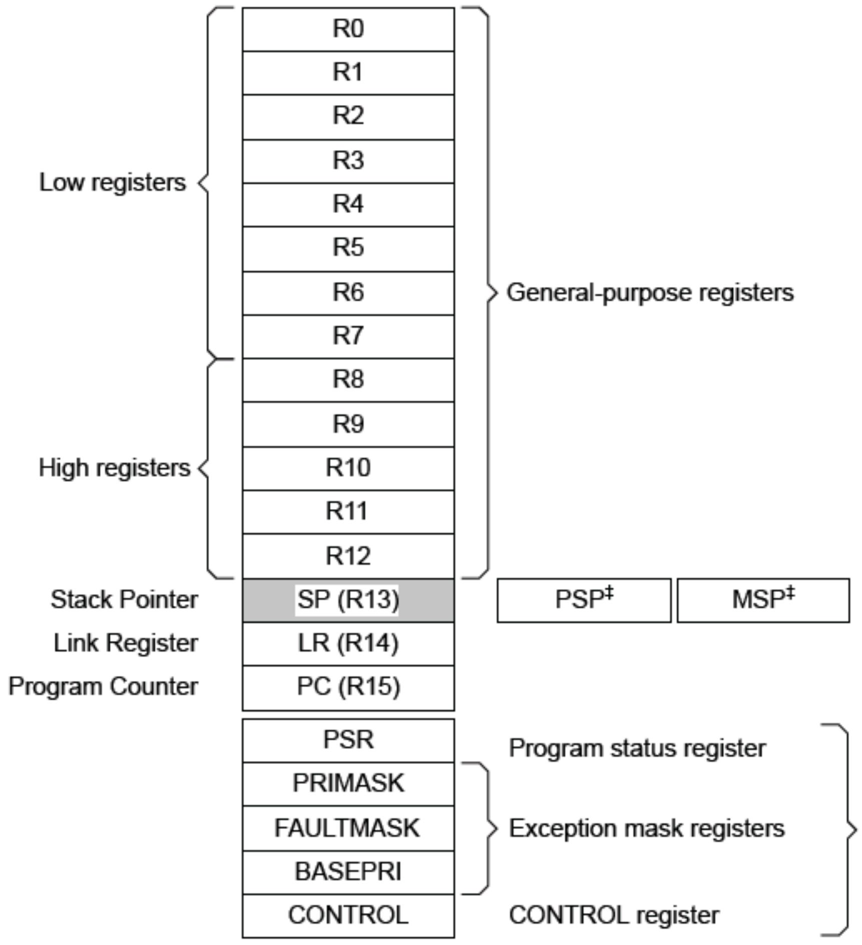
- To/From Memory & I/O Bus

CPU/Memory Structure



LM3S1968 ARM Register Set

Figure 2-3. Cortex-M3 Register Set



Variable Allocation

```
             .thumb                 ; Assemble into Thumb instructions

             .data                  ; Allocate variable in the SRAM section

W:           .int      145
X:           .int      32
Y:           .int      54
Z:           .int      97
Name:        .string      "Gary J. Minden"

;     End  of variable allocation
```

Variable References

```
;
;       Main function
;
                .text
                .clink
                .thumb
                .global     main
;
;       Addresses of variables
;
W_Adr:    .word     W
X_Adr:    .word     X
Y_Adr:    .word     Y
Z_Adr:    .word     Z
Name_Adr: .word     Name
```

Program

```
;
;      Compute Z = W + X * Y;
;
;      Load variable addresses and variable values
;
main:
             LDR       R0,W_Adr      ; Load address of W
             LDR       R0,[R0,#0]       ; Load W value

             LDR       R1,X_Adr      ; Load address of X
             LDR       R1,[R1,#0]       ; Load X value

             LDR       R2,Y_Adr      ; Load address of Y
             LDR       R2,[R2,#0]       ; Load Y value

             LDR       R3,Z_Adr      ; Load address of Z

;            BKPT      #1            ; Break

             MUL       R5,R1,R2      ; Compute X * Y
             ADDS      R5,R5,R0;     ; Compute W + X * Y

             STR       R5,[R3,#0]       ; Store result in Z

             BX        LR            ; Return
```

Information and Telecommunication Technology Center

KU THE UNIVERSITY OF KANSAS

if Statement

```
if (A == B) {
    C = A;
}
```

```
        .dwpsn file "../Exmple_C_01.c",line 30,column 2,is_stmt,isa 1
          LDR       A1, $C$CON4             ; [DPU_3_PIPE] |30|
          LDR       A2, $C$CON2             ; [DPU_3_PIPE] |30|
          LDR       A1, [A1, #0]            ; [DPU_3_PIPE] |30|
          LDR       A2, [A2, #0]            ; [DPU_3_PIPE] |30|
          CMP       A1, A2                  ; [DPU_3_PIPE] |30|
          BNE       ||$C$L1||               ; [DPU_3_PIPE] |30|
          ; BRANCHCC OCCURS {||$C$L1||}     ; [] |30|
;* --------------------------------------------------------------------------*
        .dwpsn file "../Exmple_C_01.c",line 31,column 3,is_stmt,isa 1
          LDR       A1, $C$CON2             ; [DPU_3_PIPE] |31|
          LDR       A2, $C$CON3             ; [DPU_3_PIPE] |31|
          LDR       A1, [A1, #0]            ; [DPU_3_PIPE] |31|
          STR       A1, [A2, #0]            ; [DPU_3_PIPE] |31|
;* --------------------------------------------------------------------------*
||$C$L1||:
```

if-else Statement

```
        if (A <= B) {
            C = A;
        } else {
            D = (A + B + C);
        }

    .dwpsn file "../Exmple_C_01.c",line 38,column 2,is_stmt,isa 1
      LDR       A1, $C$CON4           ; [DPU_3_PIPE] |38|
      LDR       A2, $C$CON2           ; [DPU_3_PIPE] |38|
      LDR       A1, [A1, #0]          ; [DPU_3_PIPE] |38|
      LDR       A2, [A2, #0]          ; [DPU_3_PIPE] |38|
      CMP       A1, A2                ; [DPU_3_PIPE] |38|
      BCC       ||$C$L3||             ; [DPU_3_PIPE] |38|
      ; BRANCHCC OCCURS {||$C$L3||}    ; [] |38|
;* -------------------------------------*
    .dwpsn file "../Exmple_C_01.c",line 39,column 3,is_stmt,isa 1
      LDR       A1, $C$CON2           ; [DPU_3_PIPE] |39|
      LDR       A2, $C$CON3           ; [DPU_3_PIPE] |39|
      LDR       A1, [A1, #0]          ; [DPU_3_PIPE] |39|
      STR       A1, [A2, #0]          ; [DPU_3_PIPE] |39|
    .dwpsn file "../Exmple_C_01.c",line 40,column 2,is_stmt,isa 1
      B         ||$C$L4||             ; [DPU_3_PIPE] |40|
      ; BRANCH OCCURS {||$C$L4||}      ; [] |40|
;* -------------------------------------*
||$C$L3||:
```

Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

if-else Statement

```
            if (A <= B) {
                C = A;
            } else {
                D = (A + B + C);
            }

||$C$L3||:
        .dwpsn file "../Exmple_C_01.c",line 41,column 3,is_stmt,isa 1
          LDR       A3, $C$CON2            ; [DPU_3_PIPE] |41|
          LDR       A1, $C$CON4            ; [DPU_3_PIPE] |41|
          LDR       A2, $C$CON3            ; [DPU_3_PIPE] |41|
          LDR       A4, [A3, #0]           ; [DPU_3_PIPE] |41|
          LDR       A1, [A1, #0]           ; [DPU_3_PIPE] |41|
          LDR       A2, [A2, #0]           ; [DPU_3_PIPE] |41|
          LDR       A3, $C$CON1            ; [DPU_3_PIPE] |41|
          ADDS      A1, A1, A4             ; [DPU_3_PIPE] |41|
          ADDS      A2, A2, A1             ; [DPU_3_PIPE] |41|
          STR       A2, [A3, #0]           ; [DPU_3_PIPE] |41|
;* ----------------------------------------------------------------------------*
||$C$L4||:
```

for Statement

```
          for ( Idx = 0; Idx < Nbr; Idx++ {
              .
              .
              .
          }

Assume Idx in R4 and Nbr in R5

          EOR       R4,R4,R4            ; Set Idx to zero

ForTest:
          CMP       R4,R5
          BGE       ForExit             ; Exit for-loop if R4 >= R5

          <for-loop body>

          ADD       R4,R4,#1            ; Increment Idx
          B         ForTest             ; Goto for-loop test

ForExit:
```

ASM Examples

Compute: X = In_1 + (In_2 * In_3)

```
;;
;;   Allocate variables and inputs
;;
GPR_SaveArea_p    .word    GPR_SaveArea  ; Address of GPR_SaveArea
X:                .word    0             ; Result
In_1:             .word    143           ; Inputs
In_2:             .word    8
In_3:             .word    63

Compute:

                  LDR          R0,In_1   ; R0 holds first input
                  LDR          R1,In_2   ; R1 holds second input
                  LDR          R2,In_3   ; R3 holds third input

                  MUL          R1,R1,R2  ; Compute product
                  ADD          R0,R0,R1  ; Compute sum

                  STR          R0,X      ; Store result
```