SECTION 1: Stack

Stack Example

1. Below is a snippet of code and structures at the beginning, before the code is executed. Walk through the code, and give the state of the structures at the end by filling out the blank tables.

Code snippet:
push r16
pop r17
pop r18
push r19
push r17

Structures before running code:

Register File	
r16	17
r17	200
r18	25
r19	4

Stack Pointer
4998

RAM	
4993	
4994	
4995	
4996	
4997	
4998	
4999	313
5000	200

Structures after running code:

Register File	
r16	17
r17	17
r18	313
r19	4

Stack Pointer
4997

RAM	
4993	
4994	
4995	
4996	
4997	
4998	17
4999	4
5000	200

Unsigned integer representation

- With n bits, max value that can be represented: $2^n - 1$

Binary to Decimal conversion

1	1	0	1	0	0
2^5	2^4	2^3	2^2	2^1	2^0

32 + 16 + 4

=52

6 bits, so max number possible is 2^6-1 = 63

Decimal to Binary (unsigned)

1. Find number of bits required
   $floor(\log_2 number) + 1$
2. For each bit-position, starting from highest, Repeatedly check if number greater or equal to $2^{\text{bit-position}}$,and set bit to 0 or 1 accordingly

Number of bits required (unsigned)

- **52:** $\log_2(52) = 5.7; floor(5.7) = 5; \#\text{ bits} = 6$
  - Check 1: $2^6 - 1 \geq 52; 63 \geq 52$ (YES)
  - Check 2: $2^5 - 1 < 52; 32 < 52$ (YES)
- **102:** $\log_2(102) = 6.67; floor(6.67) = 6; \#\text{ bits} = 7$
  - Check 1: $2^7 - 1 \geq 102; 127 \geq 107$ (YES)
  - Check 2: $2^6 - 1 < 102; 63 < 107$ (YES)
- **276:** $\log_2(276) = 8.10; floor(8.10) = 8; \#\text{ bits} = 9$
  - Check 1: $2^9 - 1 \geq 276; 511 \geq 276$ (YES)
  - Check 2: $2^8 - 1 < 276; 255 < 276$ (YES)

Decimal to Binary (unsigned)

52; # bits = 6

Bit positions start at 0, so 6 bits means 2^0 to 2^5-1

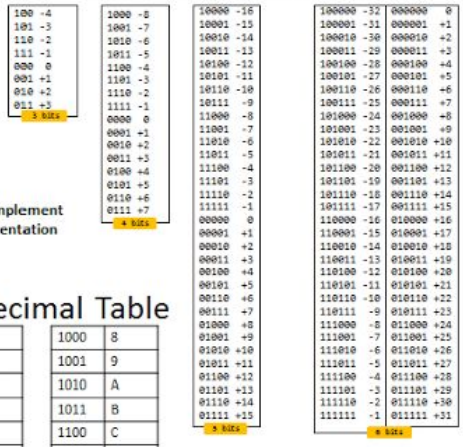
Bit position	Power of 2	Number	>=	Remainder	Bit value
5	32	52	Yes	20	1
4	16	20	Yes	4	1
3	8	4	No	4	0
2	4	4	Yes	0	1
1	2	0	No	0	0
0	1	0	No	0	0

2's complement representation

- Allows representing negative numbers
- **Arithmetic operations can be done by operating on individual bits**
- 0 is always all bits 0
- Most negative number: -2^{n-1}
- Most positive number: $+2^{n-1} - 1$

2's complement range

1 bit	This is weird: -1 to 0		
2 bits	-2	To	+1
3 bits	-4	To	+3
4 bits	-8	To	7
5 bits	-16	To	15
6 bits	-32	To	31
7 bits	-64	To	63
8 bits	-128	To	127
9 bits	-256	To	255
10 bits	-512	To	511



2's complement representation

Hexadecimal Table

0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Decimal to Binary (2's comp)

- First get number of bits
  $Floor(\log_2 abs(number)) + 2$
- If positive number, then use process we developed before and you are done
- If negative number,
  - First get representation of the absolute value
  - Then invert all bits
  - Then add +1 to the inverted bits

Decimal to Binary 2's complement

- -52
1. # bits = 7
2. Negative number
   a) Representation of +52 = 0110100

All 7 bits. Note that the MSB will always be zero in this intermediate step

   b) Invert all bits: 1001011
   c) Add +1: +0000001
   =1001100

All 7 bits. Note that the MSB will always be ONE for negative numbers at the very end

2's complement binary to decimal

- If MSB is 0, same as unsigned
- If MSB is 1, reverse steps:
  a) Invert all bits
  b) Add +1
  c) Now determine magnitude
  Remember it is a negative number

2's Complement Binary to decimal

- 1001100
- MSB is 1
  a) Invert all bits: 0110011
  b) Add +1: +0000001
  0110100
  $2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$
  = 32+16+4 = 52
  c) -52
  d) Go back to 2's complement range and check

2's complement arithmetic
It's bitwise addition!

- 52 + (-101) = -49

```
  00110100
+ 10011011
----------
  11001111
```

Extension rule: 2s complement

```
  1001100     7 bits
 11001100     8 bits
111001100     9 bits
```

- To take a number represented in X bits can get its representation in Y bits, (Y > X), copy the **MSB** into the "new" bit positions

Fixed point

- After the decimal point negative powers of 2
- 0.001

1	.	1	0	1	0
2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}

1.625

Conversion from binary to decimal

- 0.43

Power of 2	Weight	Number	number >= weight	Remainder	Bit value
2	0.5	0.43	No	0.43	0
4	0.25	0.43	Yes	0.18	1
8	0.125	0.18	Yes	0.055	1
16	0.0625	0.055	No	0.055	0
32	0.03125	0.055	Yes	0.02375	1
64	0.015625	0.02375	Yes	0.008125	1
128	0.0078125	0.008125	Yes	0.0003125	1
Represented value	0.4296875				

Floating Point Standard
IEEE-754 Standard
Single-Precision Representation

1 bit	8 bits	23 bits
S	Exponent	Fraction

- $N = -1^S * 1.fraction * 2^{exponent-127}$
  $when\ 1 \leq exponent \leq 254$
- $N = -1^S * 0.fraction * 2^{-126}$
  $when\ exponent == 0$

```
11000000110010001000000000000000
1   10000001   00100000000000000000
```

```
S = 1 ; therefore negative number
exponent = 129
fraction = 00100000000000000000000
```

$N = -1^1 * 1.1001 * 2^{129-127}$

$N = -1^1 * 1.1001 * 2^2$

$N = -1^1 * 110.01$

$N = 6.25$

743.5

0010 1110 0111.10

= 1.01110 0111.10 * 2^9

1) $fraction$ = 011100111

2) Exponent-127=9⇒Exponent = 136
10001000

3) Final representation

0 10001000 0111 0011 1000 0000 0000 000

SECTION 2: Conversions

2. Convert decimal numbers to/from unsigned magnitude and 2's complement using 8-bits by filling out the table below. Ignore the underscores, they are just there for readability.

Decimal	Signed Magnitude		2's complement	
	Binary	Hexadecimal	Binary	Hexadecimal
-113	**-113 = (-1) * (64 + 32 + 16 + 1) = 0b1111_0001**	**0xF1**	**113 = 64 + 32 + 16 + 1 = 0b0111_0001 -113 = 0b1000_1111**	**0x8F**
(-1) * (64 + 8 + 4 + 2) = -78	0b1100_1110	0xCE	**-78 = -128 + 32 + 16 + 2 = 0b1011_0010**	**0xB2**
-128 + 32 + 16 + 8 + 2 = -70	**-70 = (-1) * (64 + 4 + 2) = 0b1100_0110**	**0xC6**	0b1011_1010	0xBA

3. Convert 121.78125 from decimal to unsigned fixed point notation.

```
121/2 = 60 R1
 60/2 = 30 R0
 30/2 = 15 R0
 15/2 =  7 R1
  7/2 =  3 R1
  3/2 =  1 R1
  1/2 =  0 R1
0.78125/0.50000 = 1 R0.28125
0.28125/0.25000 = 1 R0.03125
0.03125/0.12500 = 0 R0.03125
0.03125/0.06250 = 0 R0.03125
0.03125/0.03125 = 1 R0
121.78125 = 1111001.11001
```

4. What is the approximate value of 0ECE2520 in floating point notation? You do not need to compute the exact answer, just set up the formula in decimal.

0ECE2520
= 0000_1110_1100_1110_0010_0101_0010_0000
= 0_00011101_10011100010010100100000
sign = 0
exponent = 16 + 8 + 4 + 1 = 29
fraction = .10011100010010100100000 ≈ 0.5 + 0.0625 + 0.03125 + 0.015625 + ... = 0.61051

$$(-1)^{sign} \times 2^{exponent-127} \times (1 + fraction)$$
$$\approx (-1)^{0} \times 2^{29-127} \times (1 + 0.61051)$$ **# getting to this step is full credit**
$$= 1 \times 2^{-98} \times 1.61051$$
$$= 5.08187E-30$$

5. Assuming two's complement notation, perform the following calculations using 8-bits two's complement notation. Express your answer in two's complement binary.

```
    1111
a.  10101010 (-86)
   +11111001 +(-7)
    10100011 (-93)

    11001100  (-52)
b.  -10110110 -(-74)
    00010110   (22)
```

or equivalently by taking the 2's complement and then adding

```
    1111 1
    11001100 (-52)
   +01001010 +(74)
    00010110  (22)
```

6. Compute the following boolean operations. Ignore the underscores, they are only there for visibility.

a. 1001_0101 ^ 0101_1001
   1100_1100

b. (y | ~y) & y (y is 8 bits)
   y

Instruction	Description	Op 1	Op 2	Effect on operands	Effect on specials	Example
Load/Store instructions						
ldi	Load immediate	reg 16-31	8-bit value	op1 = op2	incPC; sreg: none	ldi r16, 45
mov	Move	reg	reg	op1 = op2	incPC; sreg: none	mov r1, r2
ld	Load from RAM	reg	X	op1 = RAM[r26 + 256*r27]	incPC; sreg: none	ld r1, X
st	Store in RAM	X	reg	RAM[r26 + 256*r27] = op2	incPC; sreg: none	st X, r1
Computation Instructions						
add	Add	reg	reg	op1 = op1 + op2	incPC; sreg: NZC	add r1, r2
sub	Subtract	reg	reg	op1 = op1 - op2	incPC; sreg: NZC	sub r1, r2
inc	Increment register	reg	none	op1 = op1 + 1	incPC; sreg: NZ	inc r1
dec	Decrement register	reg	none	op1 = op1 - 1	incPC; sreg: NZ	dec r1
and	And	reg	reg	op1 = op1 & op2	incPC; sreg: NZ	and r1, r2
or	Or	reg	reg	op1 = op1 | op2	incPC; sreg: NZ	or r1, r2
eor	Exclusive-or	reg	reg	op1 = op1 ^ op2	incPC; sreg: NZ	eor r1, r2
com	Complement, Not	reg	none	op1 = ~op1	incPC; sreg: NZ	com r1
neg	Negate	reg	none	op1 = -op1	incPC; sreg: NZC	neg r1
asr	Arithmetic shift right	reg	none	op1 = op1 >> 1 MSB unmodified	incPC; sreg: NZC	asr r1
cp	Compare two registers	reg	reg	none	incPC; sreg: NZC*	cp r1, r2
subi	Subtract immediate	reg 16-31	8-bit value	op1 = op1 - op2	incPC; sreg: NZC	subi r16, 10
andi	And immediate	reg 16-31	8-bit value	op1 = op1 & op2	incPC; sreg: NZ	andi r16, 1
ori	Or immediate	reg 16-31	8-bit value	op1 = op1 | op2	incPC; sreg: NZ	ori r16, 1
adc	Add with carry	reg	reg	op1 = op1 + op2 + C	incPC; sreg: NZC	adc r1, r2
sbc	Subtract with carry	reg	reg	op1 = op1 - op2 - C	incPC; sreg: NZC	sbc r1, r2
Input/Output						
in	Read I/O register	reg	value 0-63	Read op2 into op1	incPC; sreg: none	in r1, 16
out	Write to I/O register	value 0-63	reg	Write op2 to op1	incPC; sreg: none	out 18, r1
Control-Flow						
rjmp	Relative jump	Value: -2048 to 2047	none	none	pc=pc+op1; incPC; sreg: none	rjmp 4 rjmp -3
breq	Branch if equal	Value: -64 to +63	none	none	If Z set, pc=pc+op1; incPC; sreg: none	breq 2 breq -3
brne	Branch if not equal	Value: -64 to +63	none	none	If Z clear, pc=pc+op1; incPC; sreg: none	brne 2 brne -3
brsh	Branch if same or higher	Value: -64 to +63	none	none	If C clear, pc=pc+op1; incPC; sreg: none	brsh 2 brsh -3
brlo	Branch if lower	Value: -64 to +63	none	none	If C set, pc=pc+op1; incPC; sreg: none	brlo 2 brlo -3
rcall	Call relative address	Value: -2048 to 2047	none	push pc+1	pc=pc+op1; incPC; sreg: none	rcall -10
ret	Return from subroutine	none	none		pop pc; sreg: none	ret
Stack						
push	Push register to stack	reg	none	RAM[sp] = op1 sp = sp - 1	incPC; sreg: none	push r1
pop	Pop register off stack	reg	none	sp = sp + 1 op1 = RAM[sp]	incPC; sreg: none	pop r1

```
Setting sreg
N: set if result is negative
Z: set if result is 0
add, addi, adc     C: set if op1 + op2 > 255
sub,  subi         C: set if op1 < op2 (unsigned)
neg, asr           C: set if op1 was odd
sbc                C: set if op1 + C < op2 (unsigned)
```

```
Setting sreg for cp
if (op1 > op2)  set N else clear N
if (op1 == op2) set Z else clear Z
if (op1 < op2)  set C else clear C
```

```
PIND:  IO reg 16
DDRD:  IO reg 17
PORTD: IO reg 18
SPL: IO reg 61
SPH: IO reg 62
```

```
Asm Directive   Example
labels          label_name:
byte            .byte(label_name) 0,1,1,2,3,5,8
string          .string(label_name) "hello world"
```

```
ISA Operations
lo8                     ldi r26, lo8(label_name)
hi8                     ldi r27, hi8(label_name)
```

Program layout

Prog memory

```
ldi r31, 91
out 62, r31
ldi r31, 136
out 61, r31      ---> setsp
rjmp prog_beg    ---> jump to program start
code for functions
code for functions
code for functions
code for functions
code for functions
program_beg:
code for main prog.
code for main prog.
code for main prog.
code for main prog.
code for main prog.
```

```
if (x < y)
  foo
else
  bar
rest of program
```

```
;x in r1,y in r2
cp r1, r2
brlo foo_code
bar line 1
bar line 2
bar line 3
rjmp merge_point
foo_code:
foo line 1
foo line 2
foo line 3
merge_point:
rest of prog.
```

```
if (x == y)
;x in r1,y in r2
cp r1, r2
breq foo_code
```

```
if (x ==10)
;x in r1,y in r2
ldi r31, 10
cp r1, r31
breq foo_code
```

```
if (x >= y)
;x in r1,y in r2
cp r1, r2
brsh foo_code
```

```
write reg 2 to output
ldi r31, 255
out 17, r31
out 18, r2
```

SECTION 3: Code Snippets

7. Translate the following piece of code from Python to AVR assembly.

```
x = 20
y = 30
n = 1
s = 0
while(n<5):
   s = s + n
   if(x<=y):
      x = x + y
   else:
      y = x - y
   n = n + 1
```

```
; r16 is used for x
; r17 is used for y
; r18 is used for n
; r19 is used for s
; up to you to figure out the rest of the register usage
ldi r16, 20            ; r16 = x = 20
ldi r17, 30            ; r17 = y = 30
ldi r18, 1             ; r18 = n = 1
ldi r19, 0             ; r18 = s = 0

whileLoopBegin:        ; label for while(n<5):
ldi r22, 5             ; which immediate value to load for compare?
cp r18, r22            ; which 2 registers to compare?
brsh whileLoopEnd      ; which label to jump to?
add r19, r18           ; s = s + n
mov r20, r17           ; which register to move/copy to r20?
inc r20
cp r16, r20            ; which 2 registers to compare?
brsh elseBlockBegin    ;    if x > y, then jump to elseBlockBegin
add r16, r17           ;     else fall through and execute if block; x =
x + y
rjmp ifElseEnd         ; jump to end of if-else block

elseBlockBegin:        ; label for else block
mov r21, r17
mov r17, r16
sub r17, r21           ; some statement(s)
```

```
ifElseEnd:              ; label for end of if-else block
inc r18                 ; n = n + 1
rjmp whileLoopBegin     ; jump to check condition of while loop again

whileLoopEnd:           ; label for end of while loop
halt                    ; halt to end program
```

8. Translate the following piece of code from Python to AVR assembly.

```
n = 20
counter = 1
sum = 0
while(counter <= n):
   sum = sum + counter
   counter = counter + 1
print(sum)

ASSEMBLY EQUIVALENT
ldi r16,20
ldi r17,1    ; r17 will store the counter
ldi r18,0    ; r18 will store the sum

begin_while:
cp r16,r17  → Which registers should be compared?
brlo end_while → Which label should we jump to in this case?
add r18,r17 → How should we perform the summation?
inc r17
rjmp begin_while → Which label should we jump to in this case?

end_while:
ldi r19, 255
out 17, r19
out 18, r18
```