CprE 288 – Introduction to Embedded Systems
Exam Review

Course Review for Exam 3

Topics in Assembly Programming

- Data Movement
  - Load immediate
  - Move between registers
  - Move between register and memory
- Logic & Arithmetic
- Control Flow
  - Test/compare register(s)
  - Choose the right branch
- Function call convention
  - Pass parameters and return values
  - Share registers between caller and callee

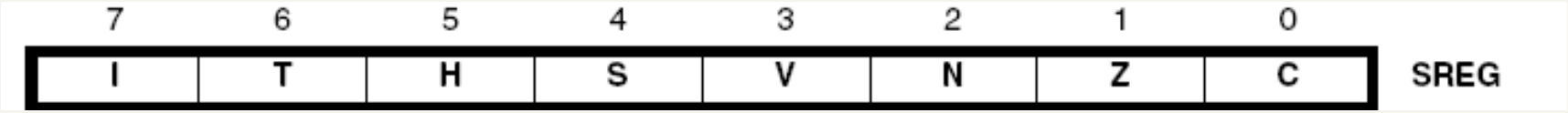
ATMEGA128: GP Registers

- 32 8-bit general purpose registers, R0-R31
  - Used for instructions to execute operations on
  - Used for accessing SRAM
  - Used for passing function parameters and return value
- What is an 8-bit register.
  - Basically just 8 D-Flips connected together

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

- Some special features to be away of
  - R16 – R31 are the only registers that can have immediate values load to them (e.g. LDI R17, 5)
  - Register pairs R27:R26, R29:R28, R31:R30 can be used as 16-bit pointers (short versions of these registers are X, Y, Z)

Status Register (SREG)



Status Register

Z: Zero flag, whether the result is zero

C: Carry Flag, whether a carry is generated

S: Sign Bit: ***Actual*** result is negative or not

V: Two's Complement Overflow Flag

N: Negative Flag, whether the result is negative or not with signed operands

H: Half Carry Flag

I: Global Interrupt Enable

T: Bit Copy Storage

Instructions to move data: Summary

- LDI Rd, K Load Immediate *Rd ← K* 1 clk
- MOV Rd, Rr Move Between Registers *Rd ← Rr* 1 clk
- MOVW Rd, Rr Move word: *Rd+1:Rd ← Rr+1:Rr* 1 clk

- LDS Rd, (k) Load Direct *Rd ← (k)* 2 clks
  - *Note: There is a ST version for each LD (except for LDI)*
- LD Rd, Y Load Indirect *Rd ← (X)* 2 clks
- LD Rd, Y+ Load Indirect & Post-Inc. *Rd←(X), X← X + 1* 2clks
- LD Rd, -Y Load Indirect & Pre-Decr. *X← X - 1, Rd←(X),* 2clks
- LDD Rd, Y+q Load Indirect + offset. *Rd ←(X+q)* 2 clks

Instructions to move data: Summary

- STS K, Rd Store Direct *Rd* ← *(k)* 2 clks
- ST Y, Rd Load Indirect *Rd* ← *(X)* 2 clks
- ST Y+, Rd Load Indirect & Post-Inc. *Rd*←*(X), X*← *X + 1* 2clks
- ST -Y, Rd Load Indirect & Pre-Decr. *X*← *X - 1, Rd*←*(X)* 2clks
- STD Y+q, Rd Load Indirect + offset. *Rd* ←*(X+q)* 2 clks

Three **indirect address (pointer) registers**: X, Y, and Z

X ⇔ R27:R26

Y ⇔ R29:R28

Z ⇔ R31:R30

LDD/STD works for Y and Z only, not X

Arithmetic Instruction

Overview of arithmetic instructions

Addition: ADD, ADC, ADIW

Subtraction: SUB, SUBI, SBC, SBCI, SBIW

Logic: AND, ANDI, OR, ORI, EOR

Compliments: COM, NEG

Register Bit Manipulation: SBR, CBR

Register Manipulation: INC, DEC, TST, CLR, SER

Multiplication1: MUL, MULS, MULSU

Fractional Multiplication1: FMUL, FMULS, FMULSU

Source: AVR Studio 4 and ATmega128: A Beginner's Guide, Page 31

HW10 Questions

```
char ch1 = 0x30;
char ch2 = 0x40;
int a = 0x1010;

ch1 = ch2;

a = ch1;
```

HW11 Questions

```
signed char ch1;
signed char ch2;
signed char flag;
int a;
int b;
signed char *pch;
int *pint;
```

```
*pch = ch1;

a = *pint;

pint = &b;

a = ch1 * ch2;

ch1 = ch1 & ch2;
```

Conditional Branches

Commonly used branches

BREQ: **EQ**ual, signed or unsigned doesn't matter

BRNE: **N**ot **E**qual, signed or unsigned doesn't matter

BRLT: **L**ess **T**han, for signed type

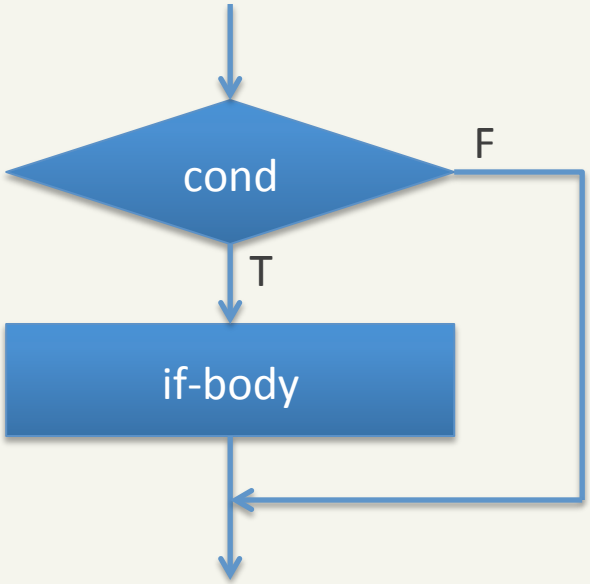
BRGE: **G**reater than or **E**qual, for signed type

BRLO: **LO**wer than, for unsigned type

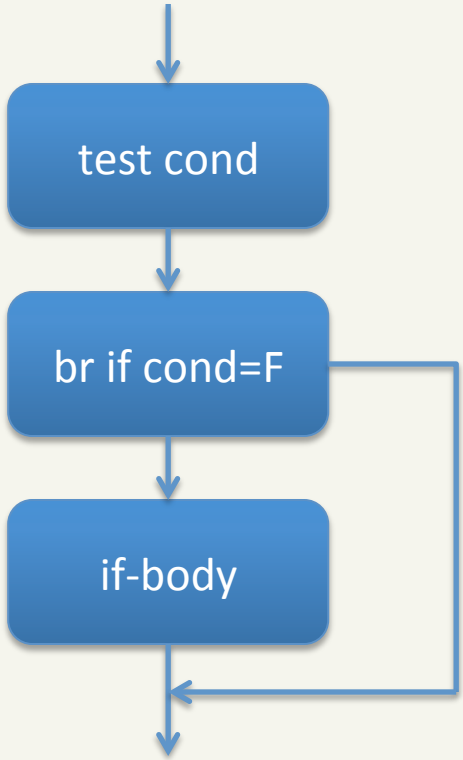
BRSH: **S**ame or **H**igher than, for unsigned type

If-Statement: Structure

Control and Data Flow Graph

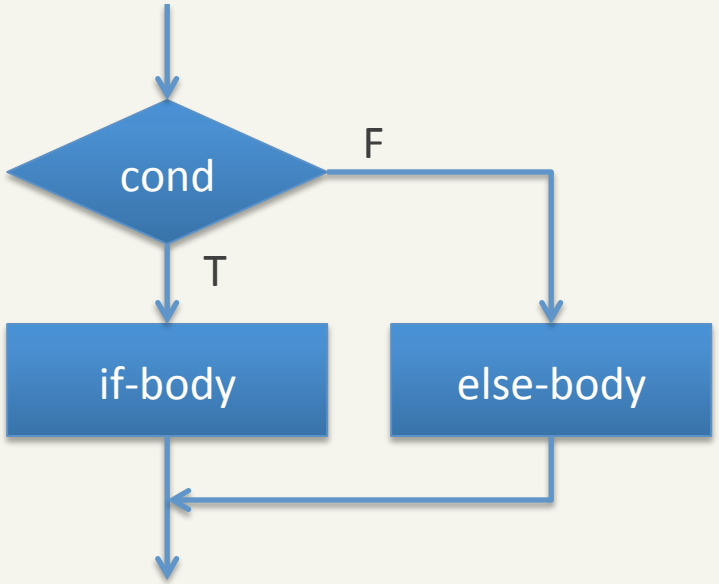


Linear Code Layout

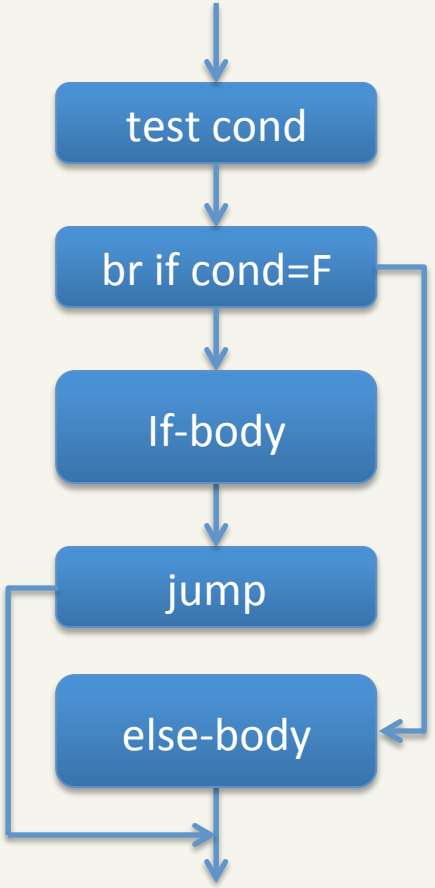


If-Else Statement: Structure

Control and Data Flow Graph



Linear Code Layout



HW12 Questions

```
extern int a, b, max;

if (a > b)
   max = a;
else
   max = b;
```

What if

```
   extern unsigned a, b, max
```

HW12 Questions

```
extern int a;
signed char flag;

if (a <= 20)
   flag = 0;
else
   flag = 1;
```

Caveats with Condition

Two types of if-conditions are trouble-free

if (a >= b)	branch if a<b, use BRLT
if (a < b)	branch if a≥b, use BRGE

What about

if (a > b)	⇔	if (b < a)
if (a <= b)	⇔	if (b >= a)
if (a > 10)	⇔	if (a >= 11)
if (a <= 10)	⇔	if (a < 11)

HW12 Questions

```
extern int a, b;
signed char flag;

if (flag)
  a = b;
else
  b = a;
```

AVR-GCC Call Convention: Parameters and Return Value

Function parameters

- R25:R24, R23:R22, ..., R9:R8
- All aligned to start in even-numbered register
  i.e. char will take two registers (use the even one)
- A long type uses two pairs
- Extra parameters go to stack

Function return values

- 8-bit in r24 (with r25 cleared to zero), or
- 16-bit in R25:R24, or
- 32-bit in R25-R22, or
- 64-bit in R25-R18

AVR-GCC Call Convention: Register Usage

How to share registers between caller and callee?

Callee-save/Non-volatile: R2-R17, R28-R29

Caller may use them for free, callee must keep their old values

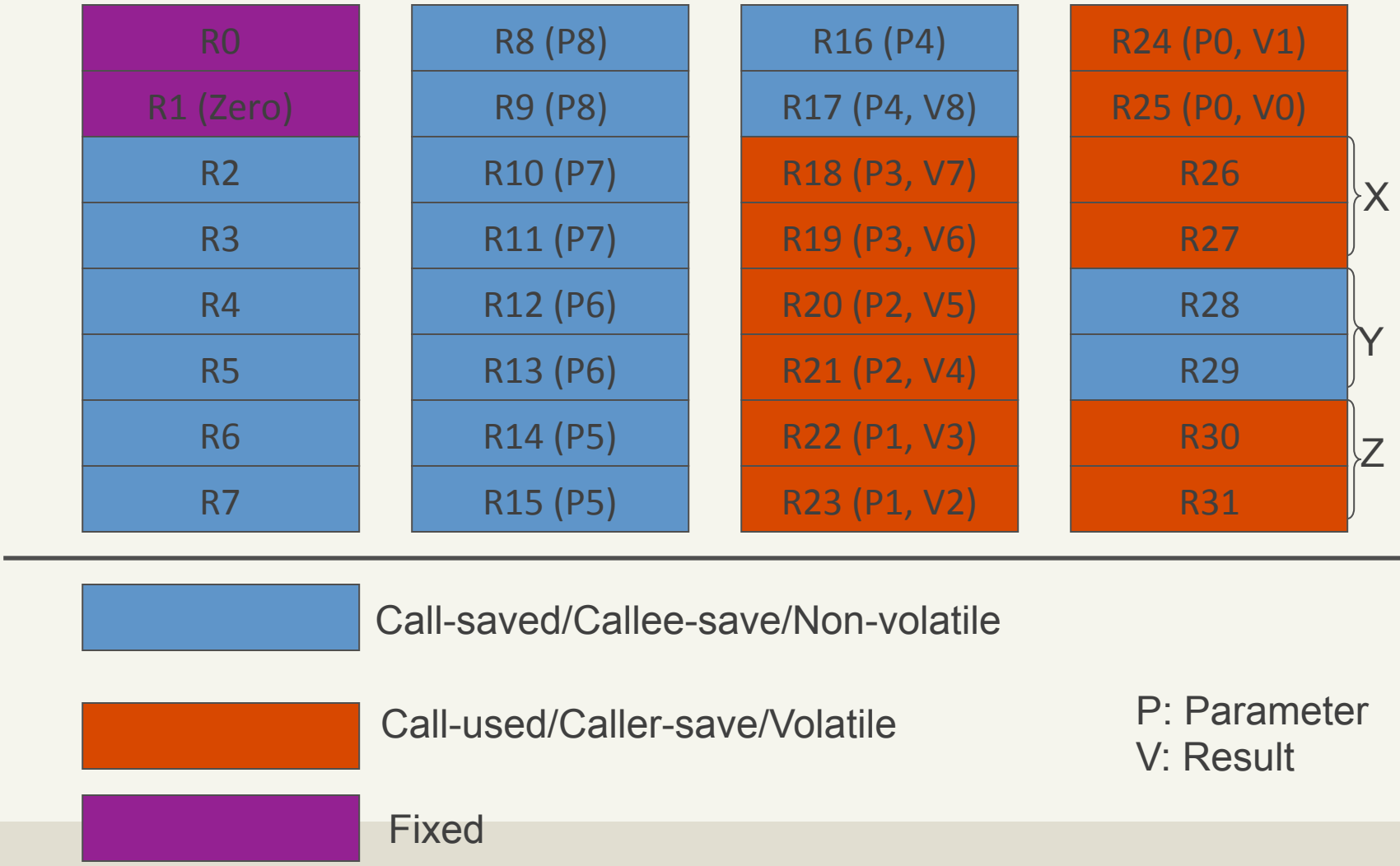
Caller-save/Volatile: R18-R27, R30-R31

Callee may use them for free, caller must save their old values if needed

Fixed registers

- R0: Temporary register used by gcc (no need to save)
- R1: Should be zero

AVR-GCC Call Convention



HW12 Questions

```
// Return the max of two integers
unsigned max(unsigned a, unsigned b);
```

HW12 Questions

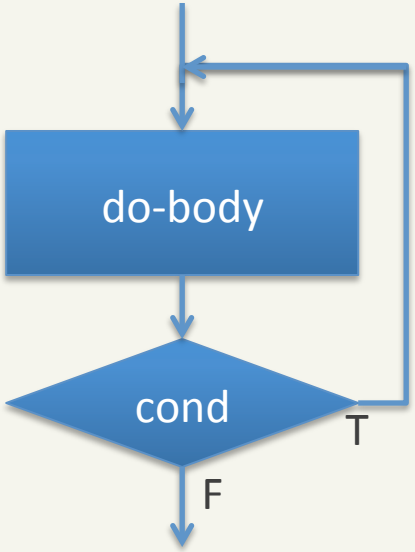
```
// Copy the contents of array X into array Y.
// Both arrays have N elements.
void copyArray(int X[], int Y[], int N);
```

HW12 Questions

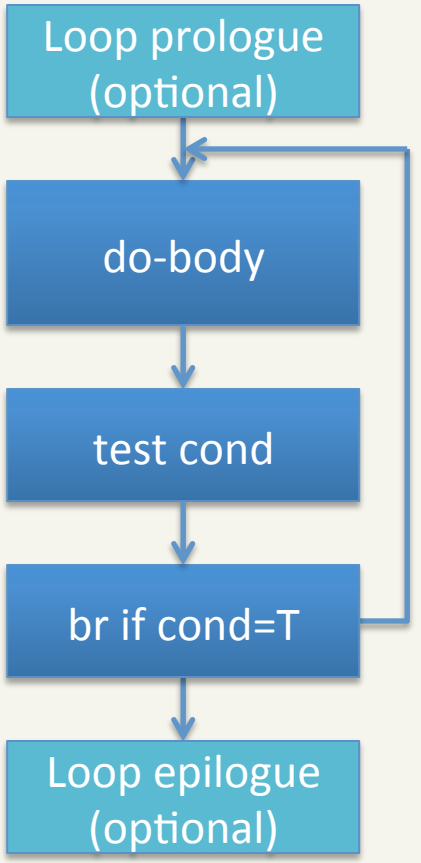
```
// Find out the maximum value of an
// array, return the value. The array
// has N elements
int maxOfArray(int X[], int N);
```

DO-WHILE Loop

Control and Data Flow Graph

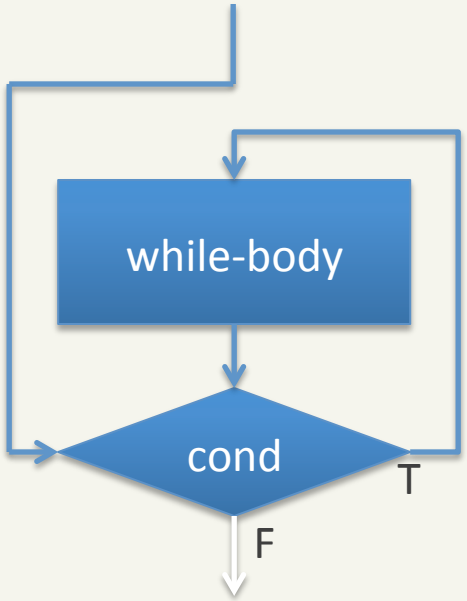


Linear Code Layout

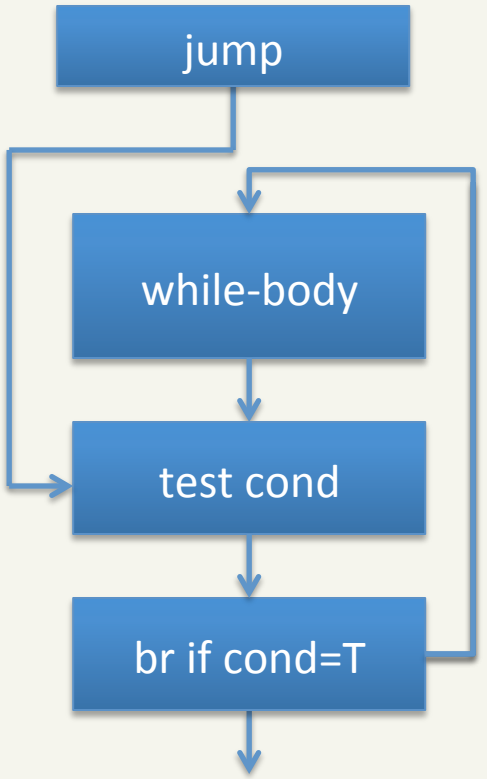


WHILE Loop

Control and Data Flow Graph



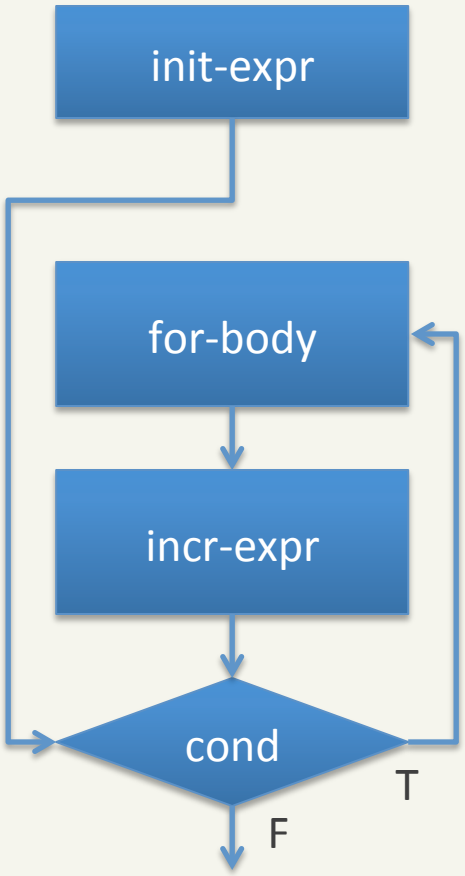
Linear Code Layout



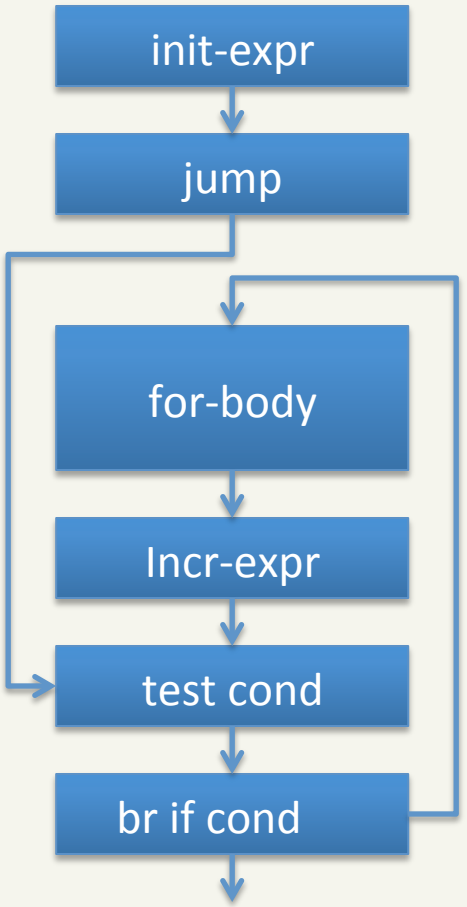
(optional prologue and epilogue not shown)

FOR Loop

Control and Data Flow Graph



Linear Code Layout



(optional prologue and epilogue not shown)