CSSE2010

CSSE2010/CSSE7201
Lecture 15

Flow of Control

School of Information Technology and Electrical Engineering
The University of Queensland

Outline

- Flow of control
  - Branching
  - Procedure Calls
  - Use of stacks
- Introduction to AVR Timer/Counters

From Previous Lectures

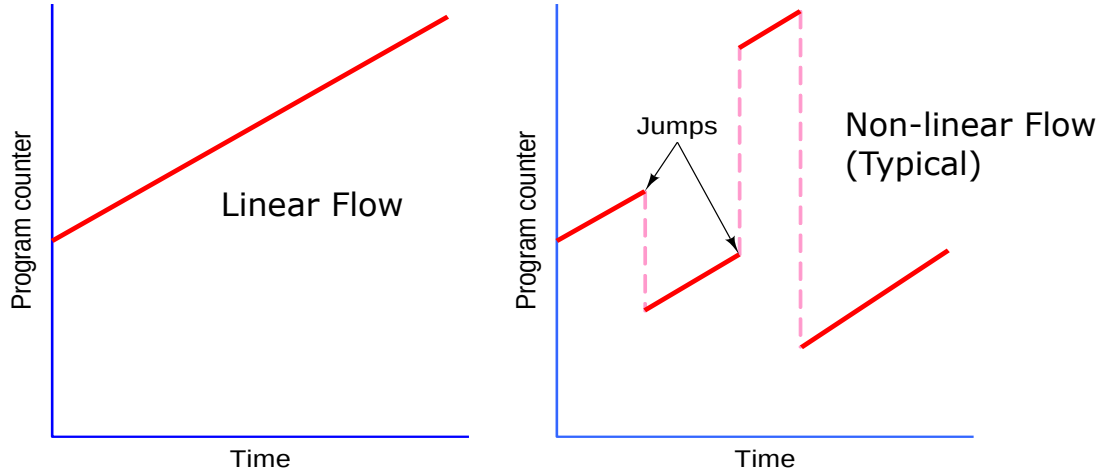
By now you should have a reasonable understanding of:

- **Atmel AVR CPU – the ALU, general purpose and special purpose registers**
- **Main hardware components of ATmega324A/ATmega328 microcontroller – AVR CPU, Data Memory, Program Memory, GPIO Ports, Timers/Counters, PWM, Communication Interfaces etc**
- **AVR Assembly language instructions**

AVR Assembly language ⇨ **AVR C programming** ⇨ **ALU Operations** ⇨ **Interaction with I/O**

Flow of Control

- Flow of Control
  - Sequence that instructions actually get executed in
  - Linear (i.e. successive instructions) unless we have
    - branches/jumps
    - procedure calls
    - etc



Flow of Control (cont.)

- Most instructions result in PC being incremented (sequential execution)
  - i.e. PC ← PC + 1
    - (Recall this from the Fetch, Decode, Execute cycle)
- Instructions aren't executed sequentially when we have
  - Branches
  - Jumps
  - Procedures
  - Interrupts / Traps

Branches

- AVR branch instructions are typically conditional
- Useful in implementing if-else and loops in assembly language
- breq and brne mentioned previously
- Example: “Greater than or equal to”
  - Instruction used depends on whether you’re comparing unsigned or signed (two’s complement) numbers

Branches – AVR Assembly example

AVR "S" Status Bit

Recall from earlier lectures:

- N bit = negative (1 if sign bit of result is 1)
- V bit = two's complement overflow

AVR also has 'S' bit:

- S = N⊕V
  - Sign-bit corrected for overflow
    - If overflow happens, sign bit will be wrong so instead of checking N bit, check S bit

Branch Instructions

Branch if status bit is equal to

	C	N	V	S=N⊕V	Z
0	brsh (=brcc)	brpl	brvc	brge	brne
1	brlo (=brcs)	brmi	brvs	brlt	breq

Jumps

- AVR has three jump instructions
  - `jmp label` – absolute jump, 32-bit instruction, 3 cycles
  - `rjmp lable` – relative jump, 16-bit instruction, 2 cycles
  - `ijmp` – indirect jump, 16-bit instruction, 2 cycles, jump target is given in Z register

Procedures

- **Procedure**
  - Group of instructions that performs some task
  - May have operands (**arguments** or **parameters**)
  - May produce result(s)
  - Can be **invoked** (called) from multiple places in program
    - Even within itself – **recursive** procedure
  - After invocation, control returns to the statement after the call
- Good use of procedures helps structure a program well
- Other names
  - **subroutine, function** (C, Python), **method** (Java)

Procedure Example – AVR Assembly

call label – direct subroutine call, 4 cycles, 32-bit instruction

rcall label – relative subroutine call, 3 cycles, 16-bit instruction

icall – indirect call, 3 cycles, 16-bit instruction

Procedure Issues

- After procedure finishes, control returns to the statement after the call
  - How does the procedure know where to return to?
- How do we specify operands (i.e. pass arguments)?
- How do we return the result?

Return Address

- Where can we store the return address?
  - Single fixed memory address or register for all procedure calls
    - BUT, procedures can't call other procedures
  - Memory location per procedure
    - Doesn't allow for recursion
  - Use a stack …

Procedure Call – Use of Stack

- Example

- Stack pointer
  - Register that keeps track of top of stack
  - AVR: SPH:SPL
    - 2 I/O registers for 16-bit stack pointer

Assembly Programs
Stack Initialisation

- Assembly language programs must initialise the stack before using it

```
.def temp=r16
    ldi temp,low(RAMEND)
    out SPL, temp
    ldi temp,high(RAMEND)
    out SPH, temp
```

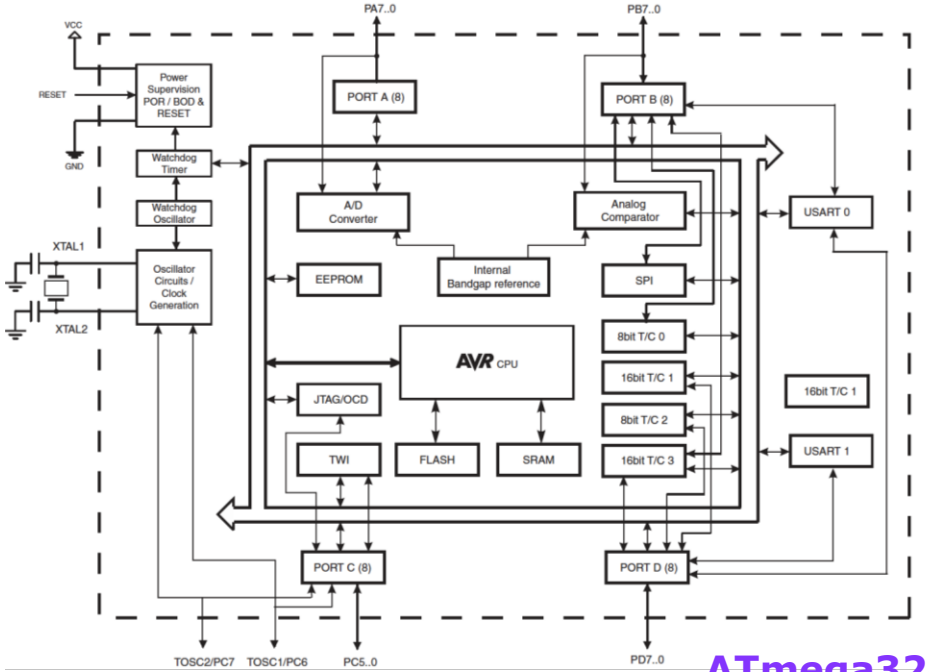
Procedure Call – Arguments and Results

- How do we pass arguments to procedures?
- How do we return results from procedures?

- Use registers…
- Or use stack

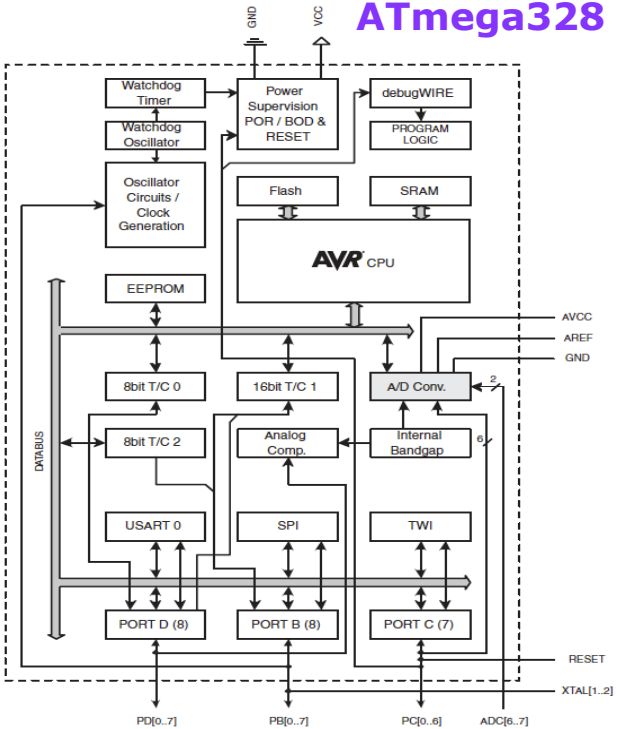
AVR Introduction to Timers/Counters

- Counter = binary up/down counter clocked on some event
- Timer = counter clocked by regular clock



ATmega324A

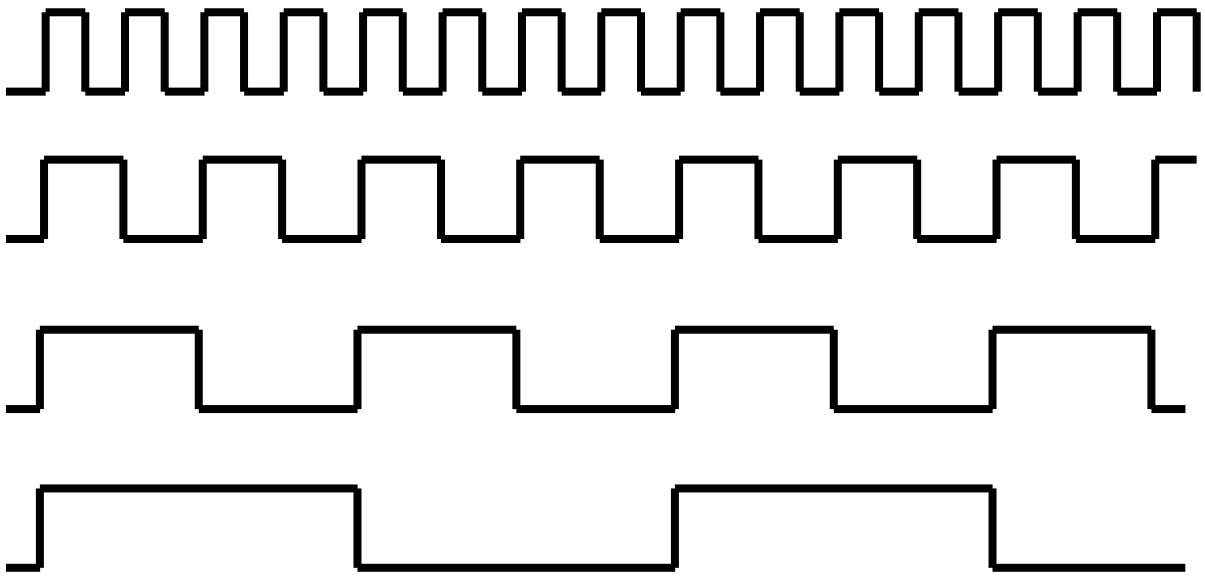
ATmega328



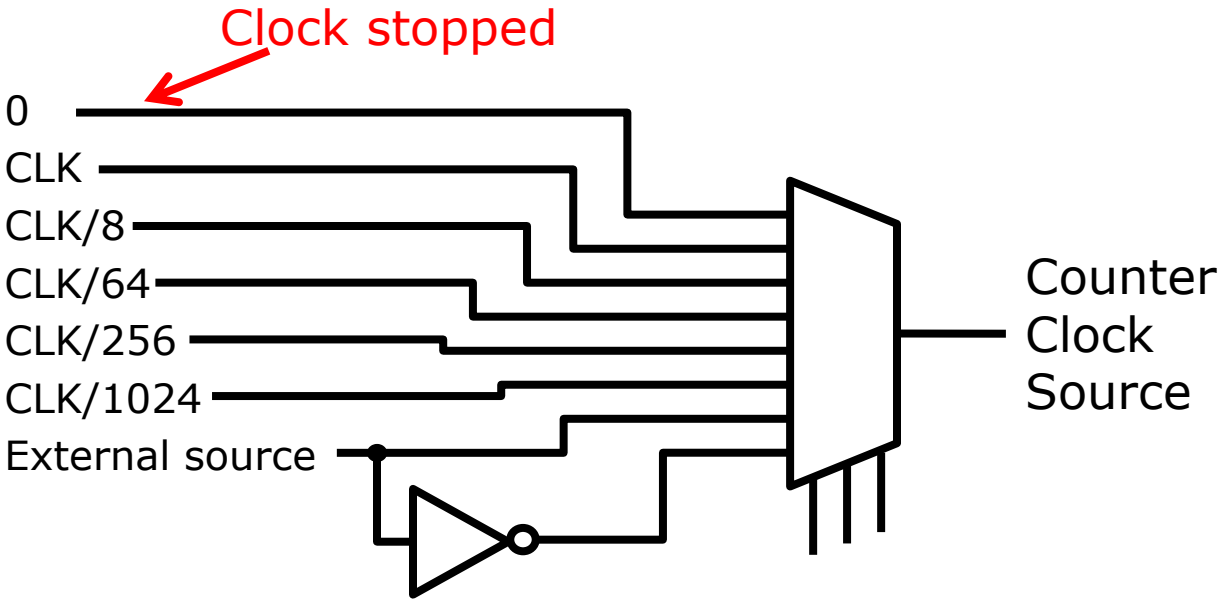
ATmega324A – Timer/Counter Clock sources

- 3 timer/counters
  - **0**: 8 bit (0 to 255)
  - **1**: 16 bit (0 to 65535)
    - Clock sources: STOPPED, CLK, CLK/8, CLK/64, CLK/256, CLK/1024, external pin rising edge, external pin falling edge
      - CLK = system clock
  - **2**: 8 bit (0 to 255)
    - Clock sources: STOPPED, CLK, CLK/8, CLK/32, CLK/64, CLK/128, CLK/256, CLK/1024
      - CLK = system clock or external oscillator

Clock prescaling

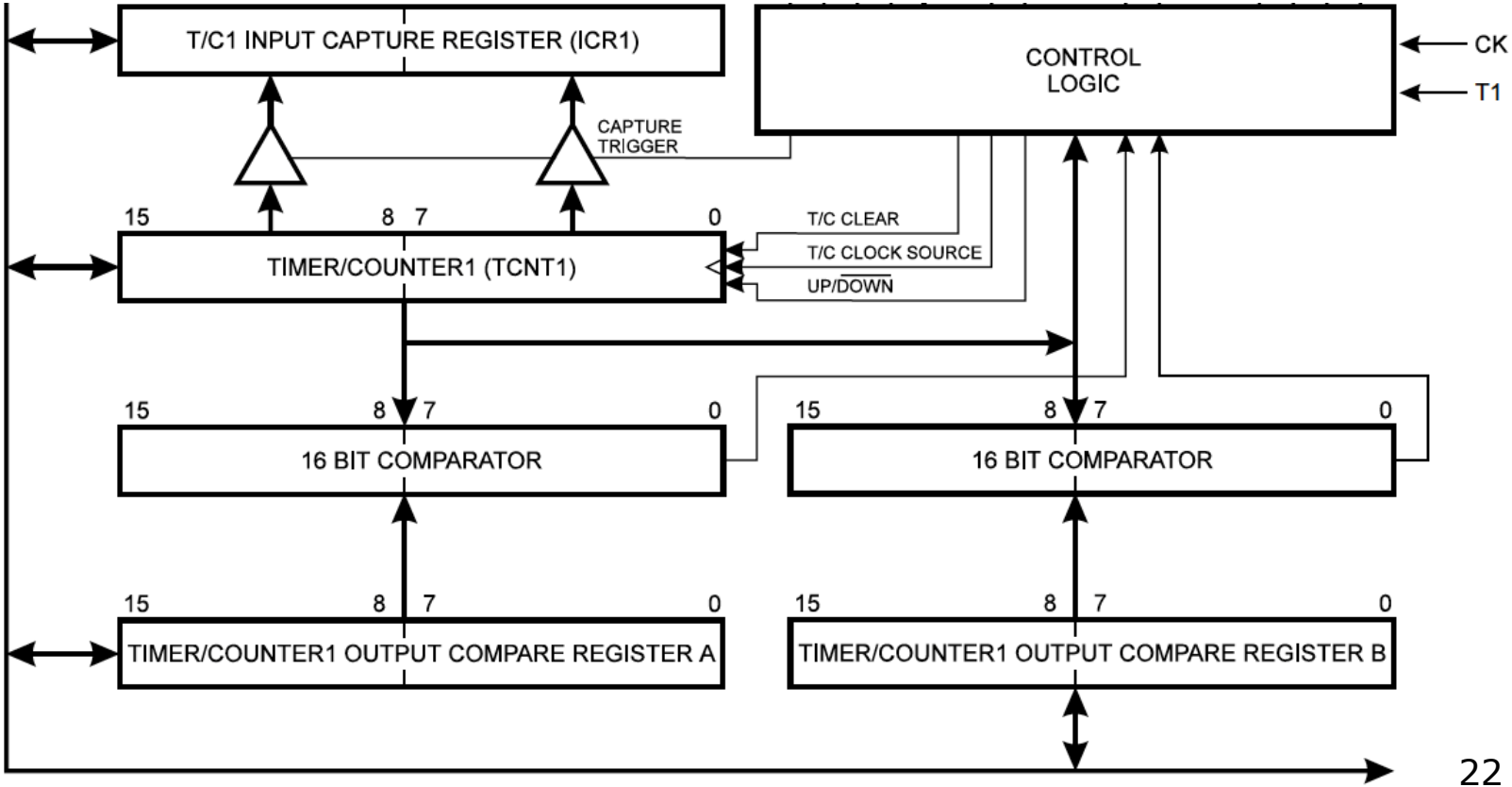


Clock Selection



Clock sources for timer/counters 0 and 1

Example 16-bit Timer/Counter



Timer/Counter Registers

- The following 8-bit I/O registers hold the current count value for each counter:
  - **TCNT0**
  - **TCNT1H**, **TCNT1L**
    - In C, can access these together as "variable" TCNT1 (a 16-bit unsigned value)
  - **TCNT2**
- You can read AND write these registers

Other Registers

- Control registers
  - Timer 0: **TCCR0A**, **TCCR0B**
  - Timer 1: **TCCR1A**, **TCCR1B**, **TCCR1C**
  - Timer 2: **TCCR2A**, **TCCR2B**
- Output compare registers
  - Timer 0: **OCR0A**, **OCR0B**
  - Timer 1:
    - **OCR1AH**, **OCR1AL** (Access in C as **OCR1A**)
    - **OCR1BH**, **OCR1BL** (Access in C as **OCR1B**)
  - Timer 2: **OCR2A**, **OCR2B**