Kent Palmkvist



TSTE12 Design of Digital Systems, Lecture 12

2022-10-10

Agenda

- Microprocessor structures and programming
- Assembly language
- C-language low level programming



2022-10-10

Practical issues

- Project presentation no later than 28/10
- I have not checked exams, may require an earlier date if exams 28/10
- Two sessions
 - 2 groups/session
- 1 group presents while others are acting as audience, then swap
- 20 minutes for each group, including demo
- Projector, DE2-board, screen, keyboard, speakers available in presentation room.



TSTE12 Design of Digital Systems, Lecture 12

2022-10-10

Microprocessor usage

- Suitable for complex programming
 - User interfaces
 - Complex state machine behavior
- Standard components
- Longer response time
 - Responses in range of us, ms, or more
- High resource utilization
 - ALU, registers etc.
- Sequential processing



2022-10-10

5

Why leave microprogrammed structures

- · Assembly language simplifies programming
 - No need to understand all small details
 - Lot of timing issues hidden
- Smaller memory footprint than microprogrammed
 - Previous microprogrammed example: long sequence of event for loading register value
 - Many control bits never used at the same time



TSTE12 Design of Digital Systems, Lecture 12

2022-10-10

Why leave microprogrammed structures, cont.

- Increase reuse
 - Architecture of processor may change while keeping the assembly language format
 - Example: 8086->80386->pentium->core2->i7
 - · Sometimes binary compatible
 - Compilers of high-level languages
 - C/C++, JAVA, Python, Perl,....



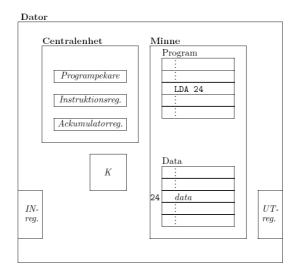
2022-10-10

2022-10-10

Model computer example

- Computer
 - Central processing
 - · Program pointer
 - · Instruction regist
 - Ackumulator
 - Memory
 - Program
 - Data
 - Peripherals
 - Inputs
 - outputs









TSTE12 Design of Digital Systems, Lecture 12

2022-10-10

Ω

General Microprocessor Structure

- Similar to microprogrammed structure
- Program information stored in memory
 - Shared with data contents
- Program counter
 - Point to next instruction to execute
- Instruction Register
 - Current executed instruction (not visible to programmer)



2022-10-10

9

Programmer model

- General Purpose Registers
 - Single or multiple registers
- Special purpose registers
 - Program counter (PC)
 - Stack pointer (SP)
 - Index registers (addressing modes)
 - Flag register
- · Memory space
 - Read or write to memory cells
 - Some addresses does not have memory cells



TSTE12 Design of Digital Systems, Lecture 12

2022-10-10 10

Microprocessor behavior

- Fetch
 - Read Program instruction from memory (pointed to by program counter (PC) register)
- Decode
 - Determine what to happen, create control signals,
- Execute
 - Update register values, move data to/from memory, arithmetic/logic operations, jumps,M



2022-10-10 11

Assembly level programming

- · Describe each instruction used to implement behavior
 - Work on internal registers and/or memory cells
- Platform dependent
 - Each processor family have their own instruction set
 - Many models of the same CPU family share instruction set (e.g., 8086 core i7)
- Maximum detail (compared to C etc.)



TSTE12 Design of Digital Systems, Lecture 12

2022-10-10 12

Instruction types

- Memory access
 - Includes I/O input and output
 - Support various addressing modes
- Arithmetic and logic
 - Modify/calculate register values
 - Include shift and rotate
- Register transfer
 - Move values between registers



2022-10-10 13

Instruction types, cont.

- · Branch and Jump
 - Includes conditional branch/jump
- Stack, Subroutines
 - pop/push, call, return from subroutine
- Control
 - Enable/disable interrupts, hardware breakpoints etc



TSTE12 Design of Digital Systems, Lecture 12

2022-10-10 14

Adressing modes

- · Immediate
 - Data in instruction itself
- Direct
 - Address defined in instruction
- Indirect
 - Register contains address to use
- Indexed
 - Address plus offset



2022-10-10 15

Assembly program example

.include "nios_macros.s"

 Calculate the sum of products

```
.global _start
start:
            movia r2, AVECTOR /* Register r2 is a pointer to vector A */ movia r3, BVECTOR /* Register r3 is a pointer to vector B */
            movia r4. N
            Idw r4, 0(r4)
                                           /* Register r4 is used as the counter for loop iterations */
            add r5, r0, r0
                                           /* Register r5 is used to accumulate the product */
LOOP:
            ldw r6, 0(r2)
                                           /* Load the next element of vector A */
            Idw r7, 0(r3)
                                           /* Load the next element of vector B */
            mul r8, r6, r7
                                           /* Compute the product of next pair of elements */
                                          /* Add to the sum */
/* Increment the pointer to vector A */
            add r5, r5, r8
            addi r2, r2, 4
                                           /* Increment the pointer to vector B */
            addi r3, r3, 4
            subi r4, r4, 1
bgt r4, r0, LOOP
                                           /* Decrement the counter */
                                           /* Loop again if not finished */
            stw r5, DOT_PRODUCT(r0) /* Store the result in memory */
STOP:
            br STOP
                              6 /* Specify the number of elements */
AVECTOR: .word
                              5, 3, -6, 19, 8, 12 /* Specify the elements of vector A */
BVECTOR: .word
                              2, 14, -3, 2, -5, 36 /* Specify the elements of vector B */
DOT_PRODUCT: .skip 4
```



TSTE12 Design of Digital Systems, Lecture 12

2022-10-10

Assembly results art

- Translate
- Indicate value in each memory adress

```
Translate 7 0010 34008001 6 0008 34008000 6 0400021 7 0 0400021 7 0 0400021 7 0 0400021 7 0 0400021 7 0 0400021 7 0 0400021 7 0 0400021 7 0 0400021 7 0 0400021 7 0 0400021 7 0 0400021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0 040021 7 0
                                                                                                                                                                                                                                                    movia r2, AVECTOR
                                                                                                                                                                                                                                                                                                                                                                  /* Register r2 is a pointer to vector A */
                                                                                                                                                                                                                                                    movia r3, BVECTOR
                                                                                                                                                                                                                                                                                                                                                                   /* Register r3 is a pointer to vector B */
                                                                                                                                                                                                                                              STOP:
                                                                                                         21
22 0048 06000000
                                                                                                                                                                                                                                                                                                                                                                    /* Specify the number of elements */
                                                                                                                                                                                                              .word 6
AVECTOR:
.word 5, 3, -6, 19, 8, 12
                                                                                                         23
24 004c 05000000
                                                                                                                                                                                                                                                                                                                                                                   /* Specify the elements of vector A */
                                                                                                                                             FAFFFFF
                                                                                                                                             13000000
08000000
                                                                                                                                                                                                              BVECTOR: .word 2, 14, -3, 2, -5, 36
                                                                                                                                                                                                                                                                                                                                                          /* Specify the elements of vector B */
                                                                                                         26 0064 02000000
                                                                                                                                             FDFFFFF
                                                                                                                                             FBFFFFFF
                                                                                                                                                                                                              DOT_PRODUCT: .skip 4...
                                                                                                         28 007c 00000000
```



2022-10-10 17

Program flow

- · Very similar to microcode
 - Single sequential execution of instructions
 - Branch/jump used to implement loops, conditional statements
- Subroutines implements function calls
 - Subroutine call saves next instructions location before jump to subroutine
 - At end of subroutine restore PC to make jump back to instruction after subroutine call



TSTE12 Design of Digital Systems, Lecture 12

2022-10-10 18

Interrupts

- Give response without polling/checking continuously
- Interrupt sequence due to external event
 - Timer, I/O, Illegal instruction, etc.
- · Interrupt routine at predefined location in memory
- · Sequence being interrupted must not notice interrupt
 - Save processor state, and restore after completed interrupt routine
 - Similar to a subroutine call, but without any instruction making the call



2022-10-10 19

C-level programming

- · Platform independent or with little platform dependence
 - Big endian vs little endian
 - Word size (8, 16, 32, 64)
- Possible to describe interrupt routines etc (same as assembly language)
- Use of hardware through memory mapped I/O
 - Store values into registers
 - Read values from registers



TSTE12 Design of Digital Systems, Lecture 12

2022-10-10 20

C-level programming, cont.

- Registers in the processor not directly accessible
 - Compiler decides where to put variables (registers, memory etc.)
- Simple constructs may be translated into long sequences of assembly code
- Less control of code
- · Possible to mix with assembly language



2022-10-10 21

I/O example

- Parallel input port for switches
 - Decode memory address, read value directly
- · Parallel output port for LED

```
Write to a
                       #define SWITCHES_BASE_ADDRESS 0x10000010
     register
                       #define LEDR_BASE_ADDRESS 0x10001000
     driving the
                       int main(void)
     LEDs
                        int * red_leds = (int *) LEDR_BASE_ADDRESS;
                                                                               /* red_leds is a pointer to the LEDRs */
Pointers
                        volatile int * switches = (int *) SWITCHES_BASE_ADDRESS;
                                                                               /* switches point to toggle switches */
 used to
                          *(red_leds) = *(switches);
                                                                               /* Red LEDR[k] is set equal to SW[k] */
 reference
                        return 0;
 memory
```



TSTE12 Design of Digital Systems, Lecture 12

2022-10-10 22

Additional subjects

- · Floating point calculations and hardware
- Caches
- · Virtual memory
- •



