Datapath and control

Prof. dr. ir. G.J.M. Smit
Dr. ir. A.B.J. Kokkeler
Processors

• Important elements
  – Combinatorial circuits
    • Operations on binary signals
  – Switches
    • Selection of 1 out of multiple
  – Memory elements
    • Registers
    • Memories
  – Clock signal
Combinatorial circuits

Input signals are presented

Collection of transistors is executing the calculation

This takes time

‘+’ indicates the arithmetic operation ‘addition’
Combinatorial circuits

Input signals are presented

Collection of transistors is executing the calculation
This takes time

After some time, the result is available
Combinatorische schakelingen

Input signals are presented

Collection of transistors is executing the calculation
This takes time

After some time, the result is available
Combinatorische schakelingen

Input signals are presented

Collection of transistors is executing the calculation
This takes time

‘*’ indicates the arithmetic operation ‘multiplication’

Except + and * there are many other operations
Switches

Diagram:
- Input 01 enters a box labeled with a plus sign (+).
- Input 10 enters a box labeled with an asterisk (*).
- The output of the plus sign is labeled 11.
- The output of the asterisk is labeled 10.
- The final output is labeled 11.
Switches

This switch has two options
=> Control uses 1 bit
ALU = Arithmetic & Logic Unit

Instruction
(0 = ‘+’
1 = ‘*’)
Memory elements

- Where do data and instruction come from?
- Where is data going to?
- Three options
  - (individual) register
  - Register banks ("registers" in AVR blockdiagram)
  - Memory banks ("memory" in AVR blockdiagram)
Register

[Diagram showing a register with 0 as input and Data out output]
Register
Register

Data in \rightarrow 0 \rightarrow 1 \rightarrow Data out
Register

At a transition from 0 to 1 on this input, “Data in” is stored and “Data out” takes over this value.

Reasoning backwards: “Data in” comes from
- Another memory element
- “outside” the processor (I/O)

Reasoning forward: “Data out” goes to
- Another memory element
- “outside” the processor (I/O)

In both cases, data can pass via combinatorial circuits, e.g. an ALU.
Register

Storage of 8 bits

At a transition from 0 to 1 on this input, “Data in” is stored and “Data out” takes over this value.
Register banks
Register banks

Switch with 4 possible positions => coded via 2 bits

Data out

00
Register banks

Switch with 4 possible positions => coded via 2 bits

Data out

01
Register banks

Switch with 4 possible positions => coded via 2 bits

Data out

10
Register banks

Switch with 4 possible positions => coded via 2 bits

Read address

Data out

11
Register banks

Switch with 4 possible positions => coded via 2 bits

0 → 1

Data in

Write address

00

Switch with 4 possible positions => coded via 2 bits

Data out

Read address

11
Register banks

Switch with 4 possible positions => coded via 2 bits

Data in

0→1

Write address

01

Read address

11

Data out
Register banks

Switch with 4 possible positions => coded via 2 bits

Data in

0 → 1

10

Write address

Switch with 4 possible positions => coded via 2 bits

Data out

26
Register banks

Switch with
4 possible positions =>
coded via 2 bits

Data in
0 → 1

Write address
11

Switch with
4 possible positions =>
coded via 2 bits

Data out
11

Read address
Register bank (RB)
Memory (MEM)

- Same functionality as Registerbank
- Read and Write not simultaneously => 1 address input + additional line that indicates read or write
- Consists of a lot more memory elements (larger address range)
Register bank (RB)

- In Atmega: Two registers can be read from the same bank
Clock signal

![Diagram showing clock signal with RB and ALU components with data inputs and outputs, addresses, and instructions.](image-url)
Clock signal

Data available for ALU

Result of ALU can be taken over

ALU busy

Clock

Data in

RB

Data out

Data in

ALU

Instruction

Read address 2

Read address 1

Write address

Read address 1

Write address
Because of the clock, time is divided into ‘timeslots’. What happens during a timeslot is determined by Read addresses 1 and 2, Write address and Instruction
Clock signal

Data available for ALU

Result of ALU can be taken over

Timeslot $i$: Read address 1, Read address 2, Instruction, Write address

Timeslot $i+1$: Read address 1, Read address 2, Instruction, Write address
Program

timeslot i: Read address 1, Read address 2, Instruction, Write address

timeslot i+1: Read address 1, Read address 2, Instruction, Write address

Example

timeslot i:  0 , 0 , 1 , 1

timeslot i+1: 1 , 0 , 0 , 2

What calculation is executed here?
Program

Example

timeslot $i$: \( \text{Read address 1, Read address 2, Instruction, Write address} \)
timeslot $i+1$: \( \text{Read address 1, Read address 2, Instruction, Write address} \)

What calculation is executed here? \( R2 = (R0*R0)+R0 \)

This way, lists of tasks (instructions) can be made that have to be executed on consecutive timeslots = Program
Program is stored in a memory = Program memory
Data available for ALU

Result of ALU can be taken over

timeslot $i$
timeslot $i+1$

Clock

Data in

RB

Data in

ALU

Data out

Program memory

Counter that is incremented each timeslot and points to the active instruction = Program counter

Read address 1, Read address 2, **Instruction**, Write address

Read address 1, Read address 2, **Instruction**, Write address

Read address 1, Read address 2, **Instruction**, Write address

Read address 1, Read address 2, **Instruction**, Write address

Read address 1, Read address 2, **Instruction**, Write address
Data available for ALU

Result of ALU can be taken over

timeslot $i$
timeslot $i+1$

Clock

Data in

RB

Data in

ALU

Data out

Read address 1, Read address 2, Instruction, Write address

Read address 1, Read address 2, Instruction, Write address

Read address 1, Read address 2, Instruction, Write address

Read address 1, Read address 2, Instruction, Write address

Program counter

10

Program counter

00

01

10

11
Program for ATmega

Data available for ALU

Result of ALU can be taken over

timeslot i

timeslot i+1

Clock

Data in

RB

Data in

ALU

Data out

address 1, address 2, Instruction

address 1, address 2, Instruction

address 1, address 2, Instruction

address 1, address 2, Instruction

Program counter

00

01

10

11

00

01

10

11

00

01

10

11
For the atmega, the order is
1. Instruction
2. Address 1 which acts as both ‘source’ and ‘destination’ address
3. ‘Source’ address 2
Program for ATmega

• Examples are based on arithmetic operations (addition, multiplication)
• Indicated as: 0 = ‘+’ en 1 = ‘*’
• In programs operations are coded by ‘Mnemonics’.
  For example ‘+’ = ADD, ‘*’ = MUL.
Program for ATmega

- Atmega can do a lot more than addition and multiplication (has more instructions and thus more hardware blocks than “RB and “ALU”)
  - Data movement
    - MOV (move): between registers
    - LD (load) and ST (store): amongst others between registers and memory
    - IN and OUT: All other addressable elements
  - Adjusting the program counter
    - JMP: fill the program counter with another value => the next timeslot, not the ‘next’ instruction will be executed but execution will proceed at another point in program memory.
  - ........
Unconditional Jump

Clock

Data in → RB → ALU → Data uit

Program counter:

- 00

Instructions:

- address 1, address 2, Instruction

- address 1, address 2, Instruction

- address 1, address 2, Instruction

- address 1, address 2, Instruction
Unconditional Jump

Data in → RB → ALU → Data out

Clock

Program counter

address 1, address 2, Instruction
00
01
10
11
Unconditional Jump

```
Data in    | RB       | ALU       | Data out

Data in   | Address 1, Address 2, Instruction 00

Data in   | Address 1, Address 2, Instruction 01

Data in   | Address 1, Address 2, Instruction 10

Data in   | Address 1, Address 2, Instruction 11

```

Clock

Program counter

10
Unconditional Jump

Program counter

RB

ALU

Data in

Data out

Clock

address 1, address 2, Instruction

address 1, address 2, Instruction

address 1, address 2, Instruction

address 1, address 2, Instruction
Unconditional Jump

At a JMP instruction:
- Program counter is not increased automatically
- Value in the address field is loaded into the Program counter
At a JMP instruction:
- Program counter is not increased automatically
- Value in the address field is loaded into the Program counter
Unconditional Jump

This way, ‘loops’ can be realized.
Conditional Jump

There are multiple Flags, each with its own condition.
Conditional Jump

- Data in to RB
- Data in to ALU
- Data out from ALU
- Zero Flag
  - ==0 → 1

Program counter: 10

<table>
<thead>
<tr>
<th>Address 1, Address 2, Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREQ</td>
</tr>
</tbody>
</table>

- Data that belongs to this instruction (CP, CPI, SUB and SUBI) is used during the BREQ instruction
Conditional Jump

Data that belongs to this instruction (CP, CPI, SUB and SUBI) is used during the BREQ instruction.
Conditional Jump

Data in → RB → ALU → Data out

Zero Flag

If Zero Flag == 0

Program counter

address 1, address 2, Instruction
address 1, address 2, Instruction
00 BREQ
address 1, address 2, Instruction

Data that belongs to this instruction (CP, CPI, SUB and SUBI) is used during the BREQ instruction.
Instruction Set Architecture

HLL program (C program) ➔ Compiler ➔ Instruction Set Architecture

Generates the binary patterns ➔ should know how these are interpreted

Describes binary patterns and the interpretation of the processor

Object code ➔ Memory

Binary patterns which can be interpreted by the processor

Processor
Instruction Set Architecture

• An alternative view

A good interface lasts through many implementations of the hardware (portability, compatibility)
What is programming?

“What programming is instructing a computer to do something for you with the help of a programming language”

Different ways to control a computer:

- High level programming language (e.g. JAVA)
  - No details of the hardware, easier to program but less efficient code
- Low-level programming language (e.g. C)
  - Big improvement over assembly language, but still close to the hardware
- Assembly (machine instructions)
A C program

#include <stdio.h>

int main(void)
{
    int a;

    a = 10;
    printf("a is now %d\n", a);

    a = a + 15;
    printf("a is now %d\n", a);

    return 0;
}

Output:

a is now 10
a is now 25
From C code to program execution (1)

C code:

```c
a = a + 15;
```

Machine code (.exe file):

```
0100 00000010
0110 00001111
0101 00000010
```

Assembler code:

```
load R2
add 15
store R2
```

Example:

```plaintext
a = a + 15;
```

Interpretation of this code is described in the ISA.

Data

execute

Together also often called compiler

Necessary but not/less interesting for this course.
Assignment

• Learning objectives
  – Design an embedded system involving both hardware and software
  – Implement the system
  – Test the system
Assignment

• Activities
  – Week 1:
    • Tuesday April 7, 2015
      – Lecture Philips
      – Lecture on Cyberphysical Systems
      – Introduction to the assignment
    • Getting familiar with the Arduino kit and development environment
    • Write a project proposal (deadline, Monday, April 13, 9:00 h.)
Assignment

• Activities
  – Week 2:
    • Tuesday April 14, 2015
      – Lecture Philips
      – Presentations by the project group: 20 minutes presentations + 10 minutes questions for each group
    • Start with execution of the project plan
Assignment

• Activities
  – Weeks 3 and 4:
    • Execution of the project plan
    • Registration of a demo on a video
    • Submit video to lecturers (deadline Friday, April 24, 23:59 h.)
Assignment

• More information on http://wwwwhome.ewi.utwente.nl/~kokkeler/ (also the ‘knight rider’ example)

• Layout Breadboard

![Breadboard Diagram]