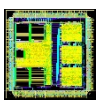
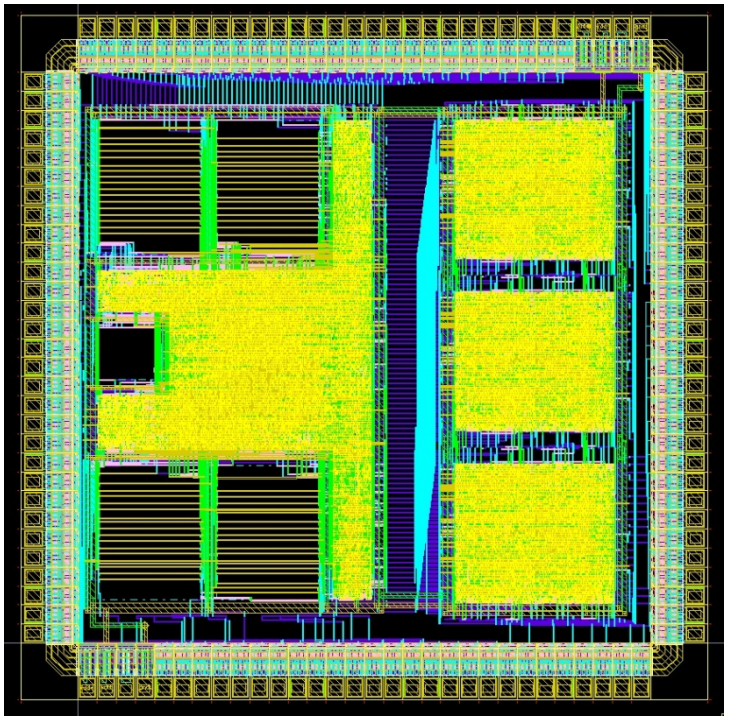
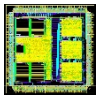


Digital Systems and Microprocessor Design (H7068)



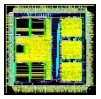
10.2. Interrupts

Daniel Roggen
d.roggen@sussex.ac.uk



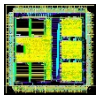
Content

- Summary: discontinuities in program flow
  - Jumps
  - Reset
- Interrupts: changing the program flow upon hardware event
- External interrupts
- Internal interrupts



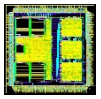
Program flow

```
PC    Adr   Inst
->    00    mov   ra,42
      02    mov   rb,3
      04    add   ra,rb
      06    cmp   ra,70
      08    ja    0Ch
      0A    jmp   04
      0C    ...
      0E    ...
```



Program flow

```
PC    Adr   Inst
      00    mov   ra,42
->    02    mov   rb,3
      04    add   ra,rb
      06    cmp   ra,70
      08    ja    0Ch
      0A    jmp   04
      0C    ...
      0E    ...
```



Program flow

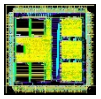
```
PC    Adr   Inst
      00    mov   ra,42
      02    mov   rb,3
->    04    add   ra,rb
      06    cmp   ra,70
      08    ja    0Ch
      0A    jmp   04
      0C    ...
      0E    ...
```

Program flow

```
PC    Adr   Inst
      00    mov   ra,42
      02    mov   rb,3
      04    add   ra,rb
->    06    cmp   ra,70
      08    ja    0Ch
      0A    jmp   04
      0C    ...
      0E    ...
```

Under “normal” circumstances PC is incremented to point to the next instruction (after each instruction / clock cycle, depending on the implementation)

In the Educational Processor PC is incremented by 1 in the fetchh and fetchl cycles

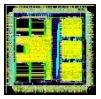


Program flow

```
PC    Adr   Inst
      00    mov   ra,42
      02    mov   rb,3
      04    add   ra,rb
      06    cmp   ra,70
->    08    ja    0Ch
      0A    jmp   04
      0C    ...
      0E    ...
```

PC is not always incremented to point to the next instruction:
- Conditional jumps
- Unconditional jumps

Here the conditional jump is not taken

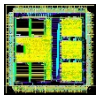


Program flow

```
PC    Adr   Inst
      00    mov   ra,42
      02    mov   rb,3
      04    add   ra,rb
      06    cmp   ra,70
      08    ja    0Ch
->    0A    jmp   04
      0C    ...
      0E    ...
```

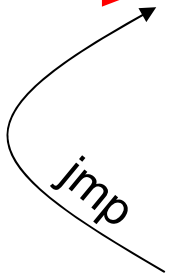
PC is not always incremented to point to the next instruction:
- Conditional jumps
- Unconditional jumps

This unconditional jump is taken

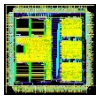


Program flow

```
PC    Adr   Inst
      00    mov   ra,42
      02    mov   rb,3
->    04    add   ra,rb
      06    cmp   ra,70
      08    ja    0Ch
      0A    jmp   04
      0C    ...
      0E    ...
```



Habitual program flow continues

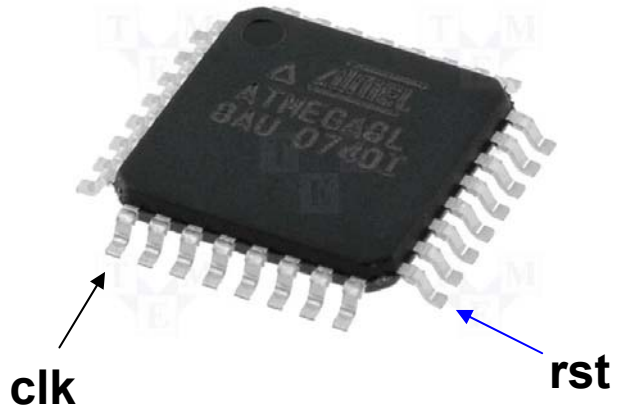


Program flow

```
PC    Adr   Inst
      00    mov   ra,42
      02    mov   rb,3
      04    add   ra,rb
->    06    cmp   ra,70
      08    ja    0Ch
      0A    jmp   04
      0C    ...
      0E    ...
```

Program flow

```
PC    Adr   Inst
      00    mov   ra,42
      02    mov   rb,3
      04    add   ra,rb
      06    cmp   ra,70
->    08    ja    0Ch
      0A    jmp   04
      0C    ...
      0E    ...
```

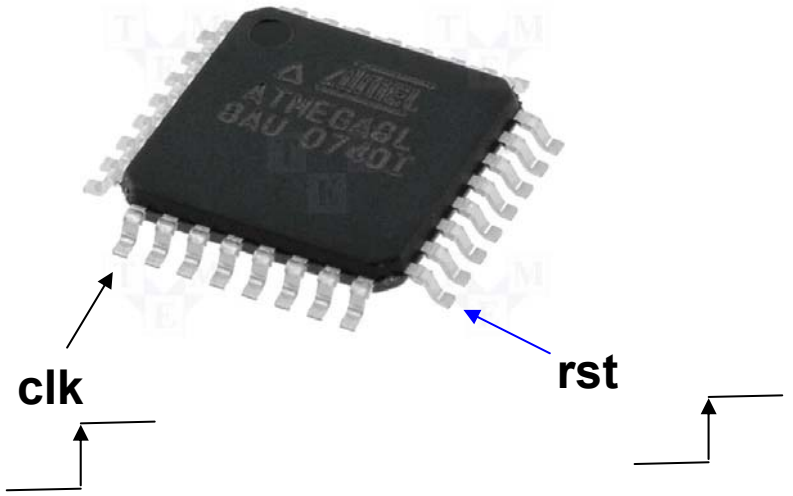


Reset creates a discontinuity in the program flow: PC set to 0

Really: more than a discontinuity as all registers are cleared – but fundamentally it is a discontinuity

Program flow

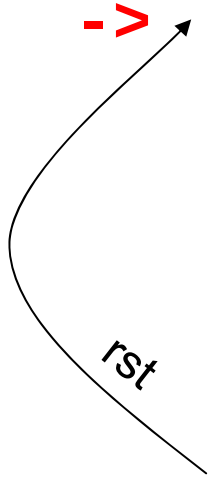
```
PC    Adr   Inst
      00    mov   ra,42
      02    mov   rb,3
      04    add   ra,rb
      06    cmp   ra,70
      08    ja    0Ch
->    0A    jmp   04
      0C    ...
      0E    ...
```

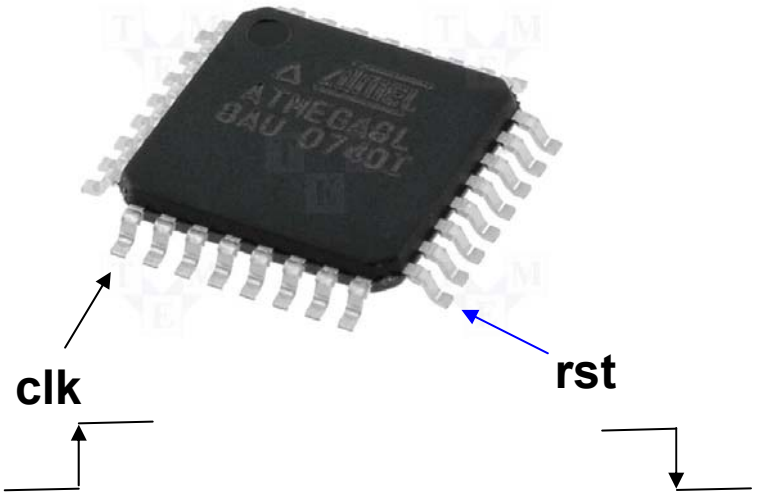


Let’s reset the processor!

Program flow

```
PC    Adr   Inst
->    00    mov   ra,42
      02    mov   rb,3
      04    add   ra,rb
      06    cmp   ra,70
      08    ja    0Ch
      0A    jmp   04
      0C    ...
      0E    ...
```

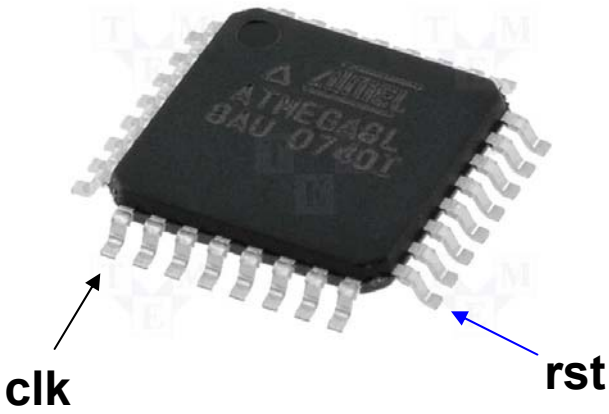




Processor after reset:
PC=0

From reset to interrupts

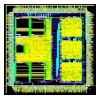
- Reset is a an external signal that changes the flow of execution



- Interrupts are another mechanism by which an external signal can change the execution flow

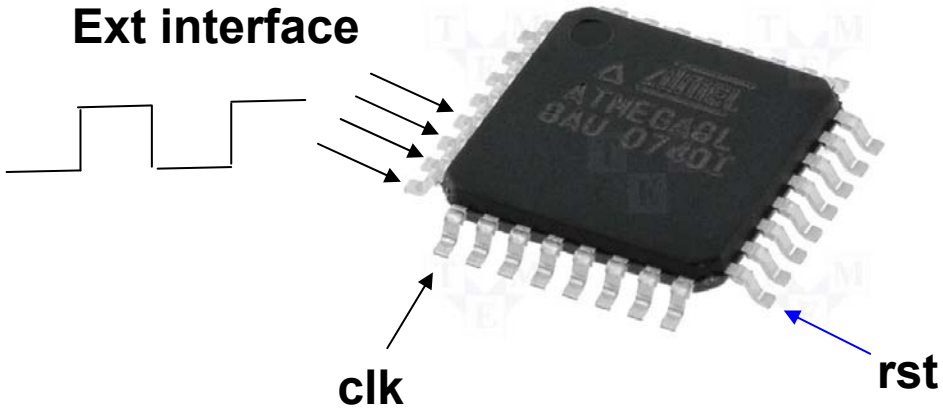
- Interrupts exist on all commercial processors and microcontrollers!

- Note: there are no interrupts in the educational processor...

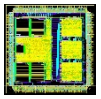


Motivation for interrupts

- Let's say a program must wait until a pin goes high on the external interface….
  - for example to count how often a pin is toggled
  - Or to implement a communication protocol in software (e.g. I2C, SPI, etc)



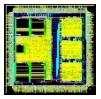
- How to implement this in a program?



Polling

- Implementation by polling: continuously reading the pin to detect changes

```
   mov  ra,0          // number of toggles
testbit:
   in   rb            // read ext input
   and  rb,1          // check bit 0
   cmp  rb,1          // bit 0 set?
   jne  testbit       //
   add  ra,1          // increment counter
                      // instead of increment we
                      // could do more complex stuff
waitclr:              // now wait until clear
   in   rb
   and  rb,1
   cmp  rb,0          // bit clear?
   jne  waitclr       // no, loop until clear
   jmp  testbit       // yes, go back to test bit set
```



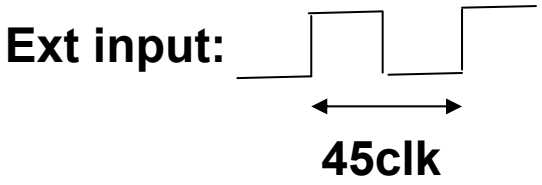
Polling

- Latency?
  - 7 instructions to react to rising edge (worst case)
  - 8 instructions to react to falling edge (worst case)

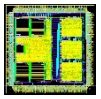
```
   mov  ra,0          // number of toggles
testbit:
   in   rb            // read ext input
   and  rb,1          // check bit 0
   cmp  rb,1          // bit 0 set?
   jne  testbit       //
   add  ra,1          // increment counter
                      // instead of increment we
                      // could do more complex stuff
waitclr:              // now wait until clear
   in   rb
   and  rb,1
   cmp  rb,0          // bit clear?
   jne  waitclr       // no, loop until clear
   jmp  testbit       // yes, go back to test bit set
```

Polling

- Latency?
  - 7 instructions to react to rising edge
  - 8 instructions to react to falling edge
- Total: 15*3=45 processor clock cycles to detect one cycle on the external input!

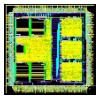


- If the external signal toggles with a period smaller than 45 clock cycles then the processor cannot count the number of toggles!



Polling

- Polling is usually a very slow solution (compared to a hardware approach) to the problem of detecting changes on input pins
  - 1/45th speed of processor in this example! (100MHz->2MHz)
  - (although some specific use cases are suitable for polling!)
- During polling the processor cannot do anything else!
- Not suitable for multitasking

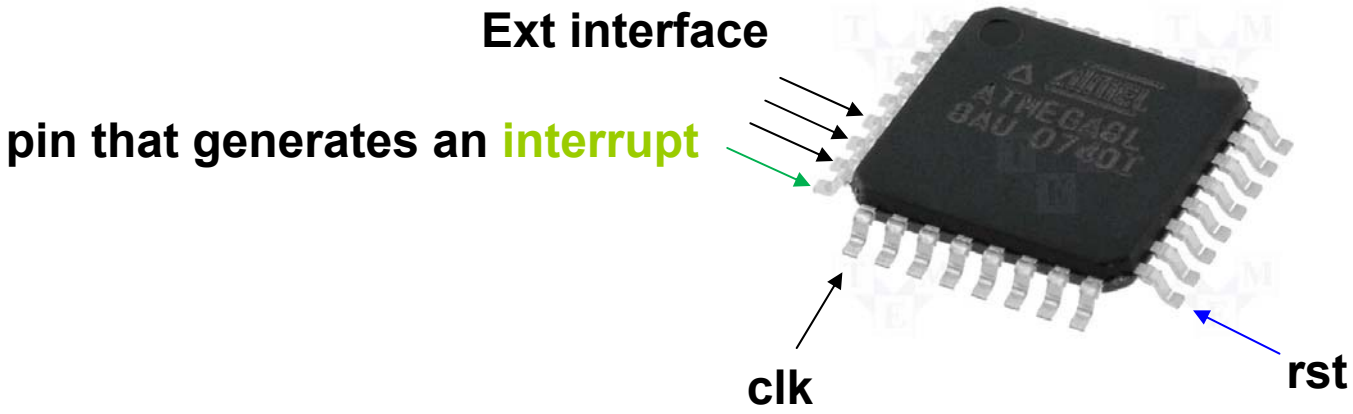


Solution 1: hardware

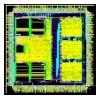
- Hardware counter (outside of processor)
- Hardware interface: see previous lectures
  - There are hardware interfaces dedicated to "count" events in many microcontrollers!
- (however that's not the point of this lecture)

Solution 2: external interrupt

- An external interrupt is a condition on a external pin that triggers an interruption of the normal program flow

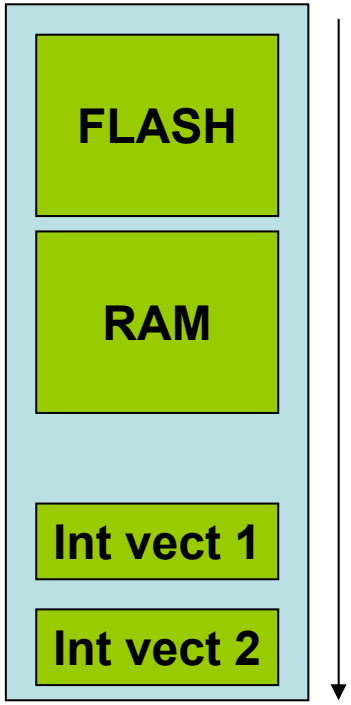


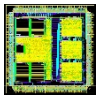
- Several conditions can trigger ints (usually configurable):
  - Rising edge
  - Falling edge
  - Any edge
  - Level-triggered



External interrupts

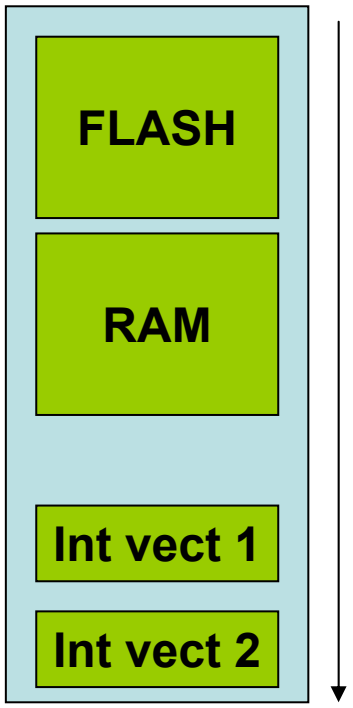
- External condition interrupts program flow
- PC set to jump to an interrupt vector
- Interrupt vector: piece of code that will handle the interrupt (do something in reaction to it)
- There can be multiple interrupts!
  - E.g. one per pin

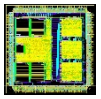




External interrupts

- The following happens during an interrupt:
- The processor keeps a return address:
  - Address of PC before the interrupt
- PC set to the interrupt vector (i.e. jump to IV)
- The processor executes the instructions in the interrupt vector
- Interrupt return instruction (RETI)
  - Indicates the end of the interrupt routine
  - The processor returns to the PC value before the interrupt (return address)

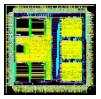




Interrupt vector table

- Processor jumps to a fixed location! (interrupt vector)
  - Typically at the start or end of memory
- Interrupt vector table (IVT): with multiple interrupts, all interrupt vectors follow each other
- Each entry is one instruction wide
- Reset can sometimes be seen as an entry in the IVT

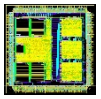
```
Adr                 What
-----               ------
00-01               First instruction on reset
02-03               Int0 (First instruction of Int0)
04-05               Int1 (First instruction of Int1)
...
0A-0B               First "normal" program instruction
```



Interrupt vector table

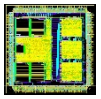
- The IVT holds only one instruction per interrupt!
  - How to have an interrupt routine of more than one instruction??
- A jump (one instruction) is stored in the IVT
- This allows to program the location of interrupt routine
  - The programmer can decide where to place the interrupt routine!
- The program continues until the RETI instruction

```
Adr                 What
-----               ------
00-01               First instruction on reset
02-03               Int0 (First instruction of Int0)
04-05               Int1 (First instruction of Int1)
...
0A-0B               First "normal" program instruction
```



IVT example

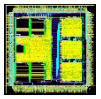
Adr	What
00-01	jmp 20
02-03	jmp 80
04-05	jmp A0
20-21	First program instruction
.....	
78-79	Last possible program instruction
80-81	First instruction for Int0
.....	
A0-A1	First instruction for Int1
.....	



Using interrupt to count toggles

- Assumption: int0 is called on rising edge of a pin

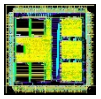
```
Adr
00                jmp 10h
02                jmp 20h

10                mov   ra,0h // number of toggles
12                ...           // do
14                ...           // something
16                ...           // continuously
18                jmp   12h     // here

20                add   ra,1h // increment counter
22                reti          // interrupt return
```



Using interrupt to count toggles

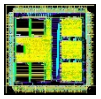
- Latency?
  - 1 instruction to go to interrupt vector
  - 1 instruction to return from interrupt vector

```
Adr
00                  jmp 10h
02                  jmp 20h

10                  mov   ra,0h // number of toggles
12                  ...         // do
14                  ...         // something
16                  ...         // continuously
18                  jmp   12h   // here

20                  add   ra,1h // increment counter
22                  reti        // interrupt return
```



External interrupts

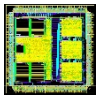
- External interrupts are a very fast approach to react to external events!
  - 2 instruction latency instead of 15 -> 7.5x speedup!
- Interrupts permit the processor to execute a main task and occasionally process "urgent" events
- Processors have an "interrupt pin" that can be used by peripherals (I/O interfaces) to trigger interrupts
  - Often used in microcontrollers
  - E.g. Interrupt once byte is sent over I2C
- Commonly used for:
  - communication
  - storage devices
  - user interaction
  - sensor sampling
- Foundation of multitasking



Possible Implementation

```
comp_ip: entity work.dffre generic map(N=>N) port
  map(clk=>clk,rst=>rst,en=>'1',d=>ipnext,q=>ip);

  ipnext <= ip+1  when fetch='1' else
                  ip when jump='0' else
                  jumpip;
```

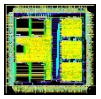
- Educational processor has no interrupts!
  - ip incremented in the fetchh and fetchl cycles
  - In exec cycle ip is unchanged if there is no jump,
  - or ip is set to the jump address if there is jump
    - unconditional jump
    - conditional jump with a valid jump condition



Possible Implementation

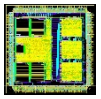
```
comp_ip: entity work.dffre generic map(N=>N) port
  map(clk=>clk,rst=>rst,en=>'1',d=>ipnext,q=>ip);

  ipnext <= ip+1  when fetch='1' else
                  ip when jump='0' and int ='0' else
                  iv when int='1' else
                  jumpip;
```

- Adding an interrupt:
  - Implemented in the same way as a jump
  - int indicates an interrupt: e.g. can be level of a signal of the external input interface
  - iv is the address of the interrupt vector (e.g. "02" in the previous example)



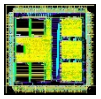
Possible Implementation

- Additionally (not shown here):
  - Keep the return address (PC before modification stored in a register or memory
  - Implement the RETI instruction (e.g. using a spare opcode)



Internal interrupts

- External interrupts react to external events (on pins)
- Internal interrupts react to processor events
- Common processor events:
  - Division by zero / invalid math operation
  - Invalid opcode
  - Data alignment check
  - etc
- For example, "invalid opcode" could be used to emulate new opcodes on an older processor!



Summary

- Interrupts change the control flow until a RETI instruction
- Multiple interrupts can placed in programmer-defined memory location using the interrupt vector table
- External interrupts allow very fast reaction to external events while allowing multitasking!
- External interrupts include pin change interrupts and peripheral interrupts
- Processors have internal interrupts to indicate issues