The Open University

T224

Computers and processors

Reference manual

Author: Mirabelle Walker

This publication forms part of an Open University course T224, *Computers and Processors*. Details of this and other Open University courses can be obtained from the Course Information and Advice Centre, PO Box 724, The Open University, Milton Keynes MK7 6ZS, United Kingdom: tel. +44 (0)1908 653231, e-mail general-enquiries@open.ac.uk

Alternatively, you may visit the Open University website at http://www.open.ac.uk where you can learn more about the wide range of courses and packs offered at all levels by The Open University.

To purchase a selection of Open University course materials visit the webshop at www.ouw.co.uk, or contact Open University Worldwide, Michael Young Building, Walton Hall, Milton Keynes MK7 6AA, United Kingdom for a brochure. tel. +44 (0)1908 858785; fax +44 (0)1908 858787; e-mail ouwenq@open.ac.uk

The Open University
Walton Hall, Milton Keynes
MK7 6AA

First published 2005. Second edition 2007.

Typeset in India by Alden Prepress Services, Chennai.

Printed and bound in the United Kingdom by Hobbs The Printers, Brunel Road, Totton, Hampshire.

SUP 964145

2.1

Contents

Important

You will be required to take this reference manual into the T224 examination with you, and when you do so it must be free of all notes and/or highlighting. Therefore do not write in it or highlight anything while you are studying the course.

1 The simulated processor

1.1 Overview

The simulated processor has a load–store, Harvard architecture.

It is essentially a 16-bit machine, in that all the general-purpose registers are 16-bit, as are the data and address buses to data memory and the address bus to program memory. The exception is the data bus between the processor and program memory, which is 24 bits wide because the instruction codes are all 24-bit.

It uses memory-mapped input–output.

It offers a two-level interrupt-handling facility.

1.2 Registers

The simulated processor has 17 registers (see Figure 1):

- 14 general-purpose registers, each 16 bits wide
- 1 instruction pointer (IP), 16 bits wide
- 1 stack pointer (SP), 16 bits wide
- 1 status register, 16 bits wide but only the least-significant 4 bits are used.

The 14 general-purpose registers are known, using the hexadecimal numbering system, as Register r1, Register r2, Register r3, up to Register rE.

Register rF can act as a fifteenth general-purpose register when the stack is not being used, but otherwise is the stack pointer.

The status register has four flags: negative flag (N); zero flag (Z); interrupt 1 flag (IF1); interrupt 2 flag (IF2). The interrupt flags are implemented only on the extended version of the simulated processor.

1.3 Instruction set

The simulated processor offers eight data-move instructions, eleven arithmetic/logic instructions and nine control instructions. It is completely symmetric across all fourteen general-purpose registers; that is, any of these fourteen registers may be used in any instruction which refers to registers.

In Tables 1 and 2, 'R' refers to a register, labelled with a suffix if more than one is referenced, and 'nnnn' refers to a 2-byte hexadecimal address or data item.

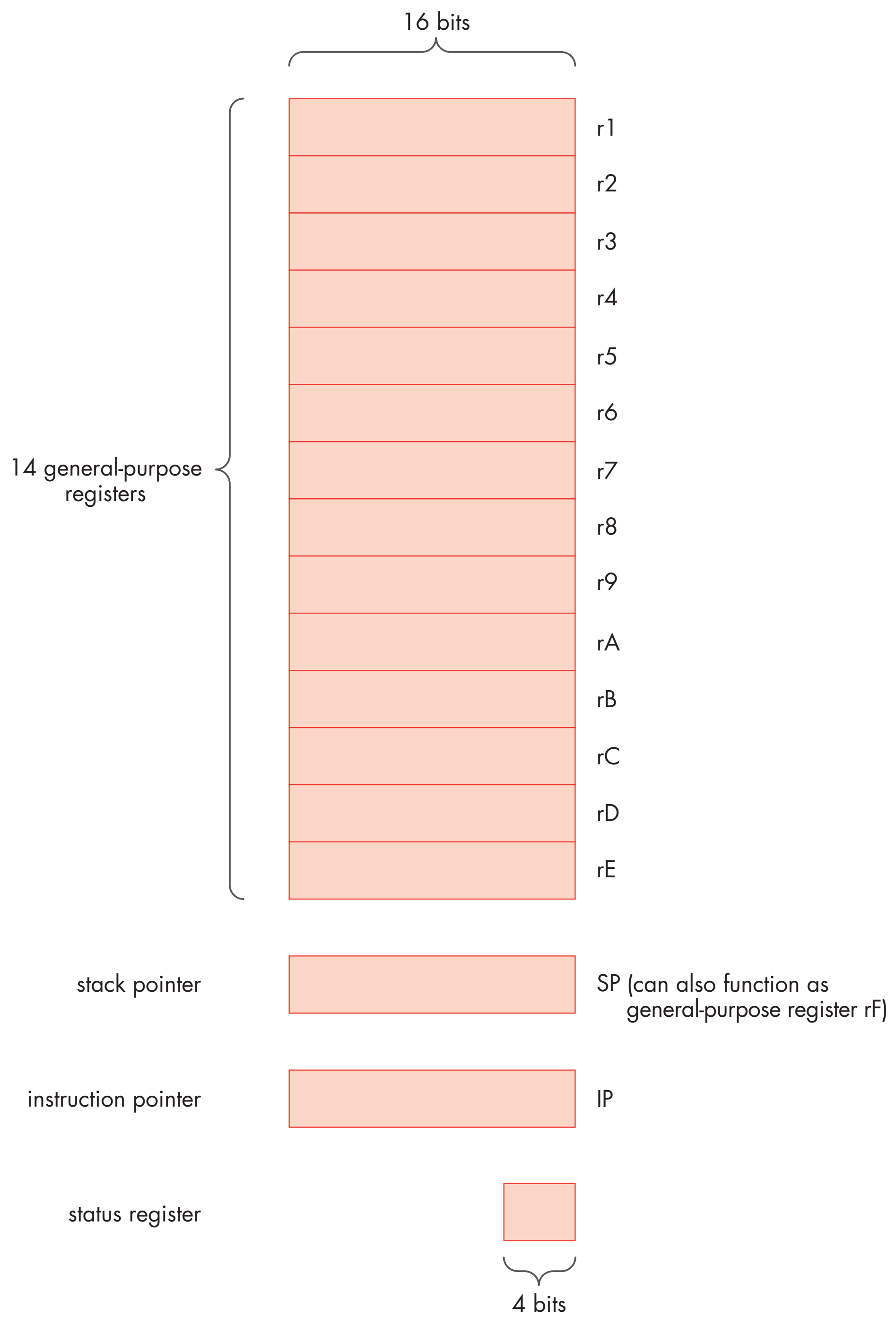


Figure 1 The programmer's model of the simulated processor

Table 1 The instruction set in alphabetical order

Instruction	**Instruction code (hexadecimal)**
add immediate	ARnnnn
add register	$020R_1R_2R_3$
AND immediate	CRnnnn
AND register	$040R_1R_2R_3$
call absolute	85nnnn
complement register	05000R
copy register	$2A00R_1R_2$
jump absolute	81nnnn
jump if negative relative	83nnnn
jump if zero relative	84nnnn
jump relative	82nnnn
load direct	ERnnnn
load immediate	DRnnnn
load register-indirect	$2E00R_1R_2$
no-op	000000
OR immediate	9Rnnnn
OR register	$010R_1R_2R_3$
pop register	26000R
push register	250R00
return	860000
return from interrupt	870000
shift left register	06000R
shift right register	07000R
stop	880000
store direct	FRnnnn
store register-indirect	$2F0R_1R_20$
subtract immediate	BRnnnn
subtract register	$030R_1R_2R_3$

Table 2 The instruction set in instruction-code order

Instruction code (hexadecimal)	**Instruction**
000000	no-op
010$R_1R_2R_3$	OR register
020$R_1R_2R_3$	add register
030$R_1R_2R_3$	subtract register
040$R_1R_2R_3$	AND register
05000R	complement register
06000R	shift left register
07000R	shift right register
250R00	push register
26000R	pop register
2A00R_1R_2	copy register
2E00R_1R_2	load register-indirect
2F0$R_1R_2$0	store register-indirect
81nnnn	jump absolute
82nnnn	jump relative
83nnnn	jump if negative relative
84nnnn	jump if zero relative
85nnnn	call absolute
860000	return
870000	return from interrupt
880000	stop
9Rnnnn	OR immediate
ARnnnn	add immediate
BRnnnn	subtract immediate
CRnnnn	AND immediate
DRnnnn	load immediate
ERnnnn	load direct
FRnnnn	store direct

1.4 Description of the instructions

add immediate

Hexadecimal instruction code

ARnnnn

Assembly language equivalent

```
add immediate rR nnnn
```

Actions

- Increase the instruction pointer's contents by 1.
- Add the value nnnn to the contents of Register rR and put the result in Register rR.
- Set the zero flag if the result is 0; otherwise clear the zero flag.
- Set the negative flag if the most-significant bit of the result is 1; otherwise clear the negative flag.

add register

Hexadecimal instruction code

020$R_1R_2R_3$

Assembly language equivalent

add register rR_1 rR_2 rR_3

Actions

- Increase the instruction pointer's contents by 1.
- Add the contents of Register rR_1 to the contents of Register rR_2 and put the result in Register rR_3 (it is permissible for R_3 to be the same as R_1 or R_2).
- Set the zero flag if the result is 0; otherwise clear the zero flag.
- Set the negative flag if the most-significant bit of the result is 1; otherwise clear the negative flag.

AND immediate

Hexadecimal instruction code

CRnnnn

Assembly language equivalent

```
AND immediate rR nnnn
```

Actions

- Increase the instruction pointer's contents by 1.
- Perform an AND operation between the value nnnn and the contents of Register rR and put the result in Register rR.
- Set the zero flag if the result is 0; otherwise clear the zero flag.
- Set the negative flag if the most-significant bit of the result is 1; otherwise clear the negative flag.

AND register

Hexadecimal instruction code

$040R_1R_2R_3$

Assembly language equivalent

```
AND register rR₁ rR₂ rR₃
```

Actions

- Increase the instruction pointer's contents by 1.
- Perform an AND operation between the contents of Register rR_1 and the contents of Register rR_2 and put the result in Register rR_3 (it is permissible for R_3 to be the same as R_1 or R_2).
- Set the zero flag if the result is 0; otherwise clear the zero flag.
- Set the negative flag if the most-significant bit of the result is 1; otherwise clear the negative flag.

call absolute

Hexadecimal instruction code

85nnnn

Assembly language equivalent

```
call absolute nnnn
```

Actions

- Increase the instruction pointer's contents by 1.
- Copy the new contents of the instruction pointer into the location in data memory whose address is in the stack pointer.
- Increase the stack pointer's contents by 1.
- Put nnnn into the instruction pointer.
- Do not change the zero or negative flags.

complement register

Hexadecimal instruction code

05000R

Assembly language equivalent

```
complement register rR
```

Actions

- Increase the instruction pointer's contents by 1.
- Complement the contents of Register rR.
- Set the zero flag if the result is 0; otherwise clear the zero flag.
- Set the negative flag if the most-significant bit of the result is 1; otherwise clear the negative flag.

copy register

Hexadecimal instruction code

$2A00R_1R_2$

Assembly language equivalent

```
copy register rR₁ rR₂
```

Actions

- Increase the instruction pointer's contents by 1.
- Copy the contents of Register rR_1 into Register rR_2.
- Do not change the zero or negative flags.

jump absolute

Hexadecimal instruction code

81nnnn

Assembly language equivalent

```
jump absolute nnnn
```

Actions

- Put nnnn into the instruction pointer.
- Do not change the zero or negative flags.

jump if negative relative

Hexadecimal instruction code

83nnnn

Assembly language equivalent

```
jump if negative relative nnnn
```

Actions

If the negative flag is set to 1

- Increase the instruction pointer's contents by 1.
- Add nnnn to the incremented contents of the instruction pointer and put the result in the instruction pointer.
- Do not change the zero or negative flags.

If the negative flag is cleared to 0

- Increase the instruction pointer's contents by 1.
- Do not change the zero or negative flags.

jump if zero relative

Hexadecimal instruction code

84nnnn

Assembly language equivalent

```
jump if zero relative nnnn
```

Actions

If the zero flag is set to 1

- Increase the instruction pointer's contents by 1.
- Add nnnn to the incremented contents of the instruction pointer and put the result in the instruction pointer.
- Do not change the zero or negative flags.

If the zero flag is cleared to 0

- Increase the instruction pointer's contents by 1.
- Do not change the zero or negative flags.

jump relative

Hexadecimal instruction code

82nnnn

Assembly language equivalent

```
jump relative nnnn
```

Actions

- Increase the instruction pointer's contents by 1.
- Add nnnn to the incremented contents of the instruction pointer and put the result in the instruction pointer.
- Do not change the zero or negative flags.

load direct

Hexadecimal instruction code

ERnnnn

Assembly language equivalent

```
load direct rR nnnn
```

Actions

- Increase the instruction pointer's contents by 1.
- Copy the contents of the location in data memory whose address is nnnn into Register rR.
- Do not change the zero or negative flags.

load immediate

Hexadecimal instruction code

DRnnnn

Assembly language equivalent

```
load immediate rR nnnn
```

Actions

- Increase the instruction pointer's contents by 1.
- Load the value nnnn into Register rR.
- Do not change the zero or negative flags.

load register-indirect

Hexadecimal instruction code

2E00R_1R_2

Assembly language equivalent

`load register-indirect via rR₁ rR₂`

Actions

- Increase the instruction pointer's contents by 1.
- Copy the contents of the location in data memory whose address corresponds to the contents of Register rR_1 into Register rR_2.
- Do not change the zero or negative flags.

no-op

Hexadecimal instruction code

000000

Assembly language equivalent

`no-op`

Actions

- Increase the instruction pointer's contents by 1.

OR immediate

Hexadecimal instruction code

9Rnnnn

Assembly language equivalent

`OR immediate rR nnnn`

Actions

- Increase the instruction pointer's contents by 1.
- Perform an OR operation between the value nnnn and the contents of Register rR and put the result in Register rR.
- Set the zero flag if the result is 0; otherwise clear the zero flag.
- Set the negative flag if the most-significant bit of the result is 1; otherwise clear the negative flag.

OR register

Hexadecimal instruction code

010$R_1R_2R_3$

Assembly language equivalent

`OR register rR₁ rR₂ rR₃`

Actions

- Increase the instruction pointer's contents by 1.

- Perform an OR operation between the contents of Register rR_1 and the contents of Register rR_2 and put the result in Register rR_3 (it is permissible for R_3 to be the same as R_1 or R_2).
- Set the zero flag if the result is 0; otherwise clear the zero flag.
- Set the negative flag if the most-significant bit of the result is 1; otherwise clear the negative flag.

pop register

Hexadecimal instruction code

26000R

Assembly language equivalent

```
pop register rR
```

Actions

- Increase the instruction pointer's contents by 1.
- Decrease the stack pointer's contents by 1.
- Copy the contents of the location in data memory whose address is in the stack pointer into Register rR.
- Do not change the zero or negative flags.

push register

Hexadecimal instruction code

250R00

Assembly language equivalent

```
push register rR
```

Actions

- Increase the instruction pointer's contents by 1.
- Copy the contents of Register rR into the location in data memory whose address is in the stack pointer.
- Increase the stack pointer's contents by 1.
- Do not change the zero or negative flags.

return

Hexadecimal instruction code

860000

Assembly language equivalent

```
return
```

Actions

- Decrease the stack pointer's contents by 1.
- Copy the contents of the location in data memory whose address is in the stack pointer into the instruction pointer.
- Do not change the zero or negative flags.

return from interrupt

This instruction is available only in the extended version of the simulated processor.

Hexadecimal instruction code

870000

Assembly language equivalent

```
return from interrupt
```

Actions

- Decrease the stack pointer's contents by 1.
- Copy the contents of the location in data memory whose address is in the stack pointer into the instruction pointer.
- Decrease the stack pointer's contents by 1.
- Copy the contents of the location in data memory whose address is in the stack pointer into the status register.

shift left register

Hexadecimal instruction code

06000R

Assembly language equivalent

```
shift left register rR
```

Actions

- Increase the instruction pointer's contents by 1.
- Shift the contents of register rR one bit to the left (towards the most-significant bit); fill the least-significant bit with 0.
- Set the zero flag if the result is 0; otherwise clear the zero flag.
- Set the negative flag if the most-significant bit of the result is 1; otherwise clear the negative flag.

shift right register

Hexadecimal instruction code

07000R

Assembly language equivalent

```
shift right register rR
```

Actions

- Increase the instruction pointer's contents by 1.
- Shift the contents of register rR one bit to the right (towards the least-significant bit); fill the most-significant bit with 0.
- Set the zero flag if the result is 0; otherwise clear the zero flag.
- Clear the negative flag.

stop

Hexadecimal instruction code

880000

Assembly language equivalent

```
stop
```

Actions

- Increase the instruction pointer's contents by 1.
- Halt execution of the processor.

store direct

Hexadecimal instruction code

FRnnnn

Assembly language equivalent

```
store direct rR nnnn
```

Actions

- Increase the instruction pointer's contents by 1.
- Copy the contents of Register rR into the location in data memory whose address is nnnn.
- Do not change the zero or negative flags.

store register-indirect

Hexadecimal instruction code

2F0$R_1R_2$0

Assembly language equivalent

store register-indirect via rR_1 rR_2

Actions

- Increase the instruction pointer's contents by 1.
- Copy the contents of Register rR_2 into the location in data memory whose address corresponds to the contents of Register rR_1.
- Do not change the zero or negative flags.

subtract immediate

Hexadecimal instruction code

BRnnnn

Assembly language equivalent

```
subtract immediate rR nnnn
```

Actions

- Increase the instruction pointer's contents by 1.
- Subtract the value nnnn from the contents of Register rR and put the result in Register rR.
- Set the zero flag if the result is 0; otherwise clear the zero flag.

- Set the negative flag if the most-significant bit of the result is 1; otherwise clear the negative flag.

subtract register

Hexadecimal instruction code

$030R_1R_2R_3$

Assembly language equivalent

`subtract register rR₁ rR₂ rR₃`

Actions

- Increase the instruction pointer's contents by 1.
- Subtract the contents of Register rR_1 from the contents of Register rR_2 and put the result in Register rR_3 (it is permissible for R_3 to be the same as R_1 or R_2).
- Set the zero flag if the result is 0; otherwise clear the zero flag.
- Set the negative flag if the most-significant bit of the result is 1; otherwise clear the negative flag.

1.5 Interrupt mechanism

The interrupt facilities are available only in the extended version of the simulated processor.

There are two levels of interrupt, Interrupt 1 and Interrupt 2. Interrupt 1 has the higher priority.

The interrupt vector for Interrupt 1 is at 0FFF and has default value 0D00. The interrupt vector for Interrupt 2 is at 0FFE and has default value 0E00. Both of these default values can be changed by the user.

Interrupt flags IF1 and IF2 in the processor's status register indicate whether Interrupt 1 and Interrupt 2 respectively are enabled. An interrupt can only be taken if it is enabled – that is, its flag is set to 1. Further, an interrupt on Interrupt 2 can only be taken if *both* interrupts are enabled.

When an interrupt occurs on Interrupt 1, then at the start of the next fetch–execute sequence the processor will automatically and in this order:

- save the status register's contents on the stack;
- clear both IF1 and IF2 to 0;
- save the instruction pointer's contents on the stack;
- set the instruction pointer to the value of the interrupt vector for Interrupt 1.

The procedure is similar for an interrupt on Interrupt 2, except that only IF2 is cleared to 0 and that the instruction pointer is set to the value of the interrupt vector for Interrupt 2.

The interrupt service routines for both Interrupt 1 and Interrupt 2 must end with a return from interrupt instruction.

When an input device causes an interrupt its status register is automatically set to 1. The program needs to clear the register to 0 after the interrupt has been serviced.

More than one device can be assigned to Interrupt 1 simultaneously. In such cases, software must be used to determine the priority of the devices within the interrupt service routine. Similarly, more than one device can be assigned to Interrupt 2 simultaneously.

2 Memory associated with the simulated processor

In the simulation, the processor has been implemented with 4096 (4K) 16-bit locations of data memory. These locations have hexadecimal addresses 0000 to 0FFF inclusive.

The simulated processor has also been implemented with 4096 (4K) 24-bit locations of program memory. These locations have hexadecimal addresses 0000 to 0FFF inclusive.

In the extended version of the simulated processor, data memory addresses 0F00 to 0FFF (hexadecimal) are reserved for the memory-mapped addresses of status and data registers relating to input and output devices.

3 Input–output associated with the simulated processor

The input–output facilities are available only in the extended version of the simulated processor.

3.1 Overview

In the simulation, the processor is implemented with the following input devices:

- a bank of 8 switches
- a 'scalepan' input
- an interrupt button

and with the following output devices:

- a bank of 8 light-emitting diodes
- a simple 4-digit 7-segment display
- a 'smart' 4-digit 7-segment display
- a buzzer
- an analogue gauge.

Table 3 shows the memory-mapped addresses for these devices' registers.

Table 3 Memory-mapped addresses for the registers in the input and output devices

Device	Register	Memory-mapped address
simple 4-digit 7-segment display	**data**	**0F00 (least-significant digit) to 0F03 (most-significant digit)**
'smart' 4-digit 7-segment display	**data**	**0F04**
buzzer	**data**	**0F05**
bank of 8 light-emitting diodes	**data**	**0F06**
analogue gauge	**data**	**0F07**
interrupt button	**status**	**0F10**
scalepan	**data**	**0F11**
scalepan	**status**	**0F12**
bank of 8 switches	**data**	**0F13**
bank of 8 switches	**status**	**0F14**

The processor offers two levels of interrupt, Interrupt 1 and Interrupt 2. Interrupt 1 has the higher priority.

3.2 Description of the input–output facilities

Bank of 8 switches (input)

By selecting the 'hardware' tab for this device, the user can set each switch independently to 'on' (up) or 'off' (down) by clicking on it.

Data register

Memory-mapped address 0F13.

Only the least-significant eight bits are used, with the least-significant bit corresponding to the rightmost switch.

1 corresponds to 'on' and 0 to 'off'.

Status register

Memory-mapped address 0F14.

The least-significant bit is set to 1 when an interrupt (see below) has occurred.

Interrupt facility

This facility can be switched on (activated) or switched off, using the 'properties' tab. The default setting is off.

When the interrupt facility is on, an interrupt is caused when any one of the switches changes its state.

Using the 'properties' tab, the switch bank's input can be set up to cause an interrupt on either Interrupt 1 or Interrupt 2. The default setting for the interrupt facility is Interrupt 1.

When an interrupt occurs, the status register is set to 0001.

Scalepan (input)

By selecting the 'hardware' tab for this device, the user can duplicate the effects of placing different weights in the scalepan by clicking with the mouse to move the scalepan up or down.

Data register

Memory-mapped address 0F11.

The contents are set to the digital equivalent of the vertical position of the scalepan, with 0000 corresponding to the highest position.

In 3-bit mode (see below) the three least-significant bits of the data register are used; in 8-bit mode the eight least-significant bits are used.

Status register

Memory-mapped address 0F12.

The least-significant bit is set to 1 when an interrupt (see below) has occurred.

Interrupt facility

This facility can be switched on (activated) or switched off, using the 'properties' tab. The default setting is off.

When the interrupt facility is on, this device causes an interrupt on Interrupt 2.

When an interrupt occurs, the status register is set to 0001.

3-bit and 8-bit working

By selecting the ‘properties’ tab, the user can choose between 3-bit and 8-bit working modes. In 3-bit mode the analogue position of the scalepan is converted into a 3-bit binary representation before being stored in the data register; in 8-bit mode the analogue position of the scalepan is converted into an 8-bit binary representation.

The default setting is 8-bit working.

Interrupt button (input)

Clicking on this button causes an interrupt on Interrupt 1.

Status register

Memory-mapped address 0F10.

When an interrupt occurs, the status register is set to 0001.

Bank of 8 light-emitting diodes (output)

Each of these lights can be independently lit by setting the corresponding bit in the data register to 1.

Data register

Memory-mapped address 0F06.

Only the least-significant eight bits are used, with the least-significant bit corresponding to the rightmost LED.

1 corresponds to ‘on’ and 0 to ‘off’.

Simple 4-digit 7-segment display (output)

Each of the segments in each 7-segment display can be independently lit by setting the appropriate bit in the appropriate data register to 1.

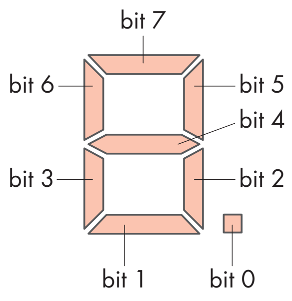


Figure 2 The relationship between the segments and the bits of the data registers

Data registers

Memory-mapped addresses 0F00 to 0F03; the base address is 0F00.

0F00 is used for the least-significant, rightmost, digit of the display and 0F03 for the most-significant, leftmost, digit.

In each of these registers, bits correspond to segments as shown in Figure 2.

‘Smart’ 4-digit 7-segment display (output)

This device takes the value in its data register and converts it to either the denary equivalent or the hexadecimal equivalent, as chosen by the user, before displaying it on the four 7-segment displays.

Leading zeros are inserted for hexadecimal displays, but not for denary ones. The error message 'Err' is displayed if denary mode has been chosen and the binary number in the data register is greater than denary 9999.

Data register

Memory-mapped address 0F04.

Denary and hexadecimal modes

These are selected by the user via the 'properties' tab. The default is denary mode.

Buzzer (output)

This device buzzes when the least-significant bit of its data register is set to 1. It is silent when the least-significant bit of its data register is cleared to 0. There is also an associated on-screen graphic.

Data register

Memory-mapped address 0F05.

Analogue gauge (output)

This device takes the value in its data register and displays it on a horizontal analogue scale.

Data register

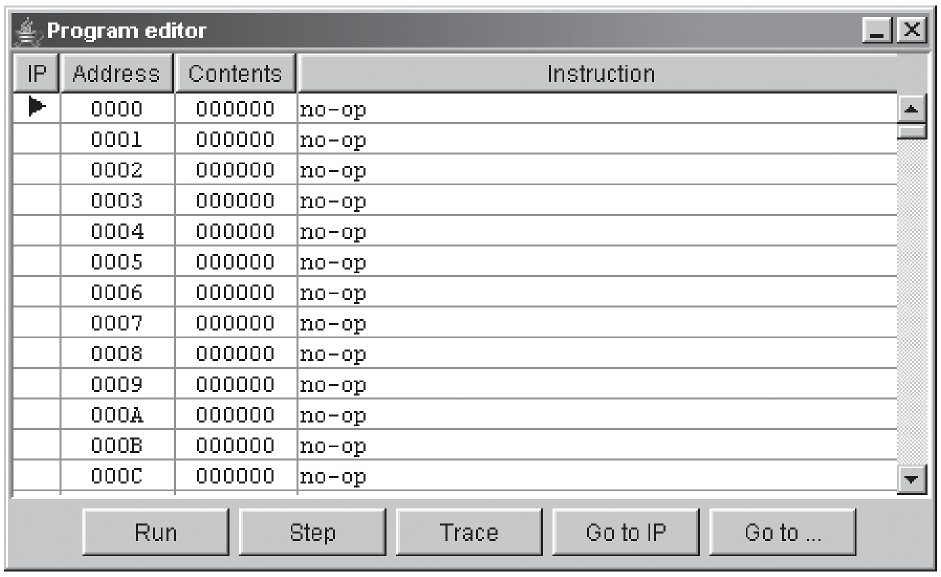
Memory-mapped address 0F07.

Only the least-significant eight bits are used, so only values from 0000 to 00FF can be displayed correctly (higher values are displayed as if they were 00FF).

4 The simulation environment

4.1 The program editor

The Program editor window is shown in Figure 3. This window can be re-sized vertically and/or moved around the screen using the mouse. A duplicate of the window can be opened via the View menu, thus enabling the user to work with two different program fragments simultaneously.


Program editor

IP	Address	Contents	Instruction
►	0000	000000	no-op
	0001	000000	no-op
	0002	000000	no-op
	0003	000000	no-op
	0004	000000	no-op
	0005	000000	no-op
	0006	000000	no-op
	0007	000000	no-op
	0008	000000	no-op
	0009	000000	no-op
	000A	000000	no-op
	000B	000000	no-op
	000C	000000	no-op

Figure 3 The Program editor window

IP field

The leftmost field of the window, IP, shows a ► symbol alongside the address corresponding to the instruction pointer's contents. By double-clicking in this field alongside a different address, the user can move the highlight and also change the instruction pointer's contents to that address.

Address and Contents fields

The next two fields of the window, Address and Contents, show the address and the contents respectively of locations in the program memory. The default contents are 000000, which correspond to the no-op instruction.

Instruction field

The rightmost field of the window, Instruction, shows the instruction corresponding to the adjacent instruction code in the Contents field. The user can change this instruction by first double-clicking on the instruction already present and then either single-clicking on items in the drop down menus (using the ▼ symbol to open the menus where necessary) or typing in a hexadecimal number, as appropriate. A double-click will terminate the entry at any time.

The instructions can be edited using the cut, copy and paste facilities in the Edit menu.

Single instructions or blocks of instructions can be deleted by first highlighting the instruction or instructions and then cutting. No-op instructions replace the cut instructions.

Single instructions can be moved from one location to another by first highlighting the instruction, then cutting, then clicking on the line where the instruction is to go, then pasting.

Blocks of instructions can be moved by first highlighting them, then cutting, then clicking on the first line where the instructions are to go, then pasting.

Individual instructions or blocks of instructions can also be copied and pasted elsewhere in program memory.

All instructions can be simultaneously changed to no-ops by selecting Reset from the Options menu and then selecting Program memory. This action clears the contents of all locations in program memory to 000000.

Run button

By clicking on the Run button at the bottom of the window, the user causes the simulated processor to execute instructions, starting with the one in the location whose address is in the instruction pointer. While the processor is running in this way, the Run button becomes a Stop button and clicking on this button stops the processor.

Step button

By clicking on the Step button at the bottom of the window, the user causes the simulated processor to execute the instruction in the location whose address is in the instruction pointer.

Trace button

By clicking on the Trace button at the bottom of the window, the user causes the simulated processor to execute instructions, starting with the one in the location whose address is in the instruction pointer. This execution is sufficiently slow that the user can follow what is happening, and the trace speed can be set to Fast, Medium or Slow from the Options menu. The default is Fast. While the processor is tracing in this way, the Trace button becomes a Stop button and clicking on this button stops the processor.

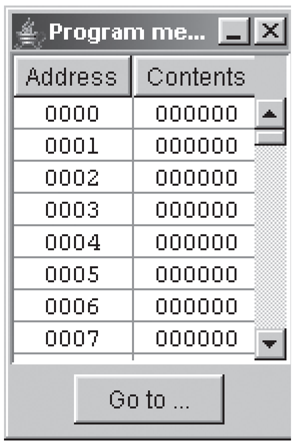
Go to IP button

Clicking on the Go to IP button at the bottom of the window causes the Program editor window to scroll to show a block of program memory which includes the location whose address is currently in the instruction pointer.

Go to button

Clicking on the Go to button at the bottom of the window opens a dialogue box which prompts the user to enter an address in program memory; the Program editor window then scrolls to show the contents of locations at and near that address.

4.2 The program memory



Address	Contents
0000	000000
0001	000000
0002	000000
0003	000000
0004	000000
0005	000000
0006	000000
0007	000000

Figure 4 The Program memory window

The Program memory window is shown in Figure 4. This window can be resized vertically and/or moved around the screen using the mouse.

The default contents of all locations in program memory are 000000. The user can return the contents of all locations to this default value by selecting Reset from the Options menu and then selecting Program memory. This action also changes all instructions in the Instruction field of the Program editor window to no-ops.

The user cannot alter the contents of individual locations in program memory via the Program memory window; all changes to individual locations in program memory must be made by entering, deleting or moving instructions in the Instruction field in the Program editor.

Clicking on the Go to button at the bottom of the window opens a dialogue box which prompts the user to enter an address in program memory; the Program memory window then scrolls to show that address.

4.3 The registers

The Registers window is shown in Figure 5. This window can be moved around the screen using the mouse.

The default contents of all registers are 0000, except for the stack pointer, whose default contents are 0800. The user can return the contents to these default values at any time by selecting Reset from the Options menu and then selecting Registers.

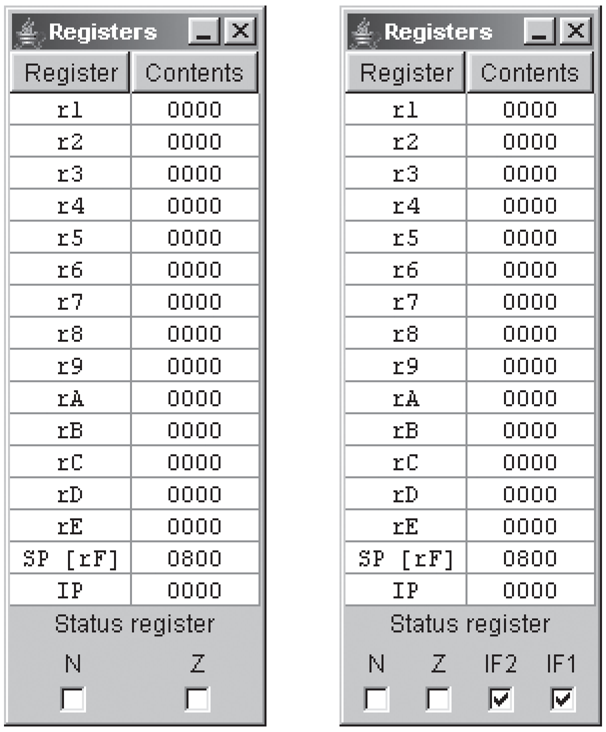
The user can alter the contents of any of Registers r1 to rE, along with the stack pointer (Register rF) and the instruction pointer, by double-clicking in the Contents field alongside them and then entering the new contents and pressing Enter.

The current contents of the status register are indicated by four flags at the bottom of the Registers window: N (negative); Z (zero); IF1 (interrupt 1); IF2 (interrupt 2). A tick indicates that the flag's value is 1; a blank indicates that it is zero. Only the negative and zero flags are shown in the basic version of the simulated processor.

Flags are set or cleared by the processor during normal program operation. The user can also set or clear flags while testing a program, by clicking under N, Z, IF1 or IF2 as appropriate.

4.4 The data memory

The Data memory window is shown in Figure 6. This window can be re-sized vertically and/or moved around the screen using the mouse. A duplicate of the window can be opened via the View menu, thus enabling the user to view two different blocks of data simultaneously.



Register	Contents
r1	0000
r2	0000
r3	0000
r4	0000
r5	0000
r6	0000
r7	0000
r8	0000
r9	0000
rA	0000
rB	0000
rC	0000
rD	0000
rE	0000
SP [rF]	0800
IP	0000

Status register

N	Z
☐	☐

Register	Contents
r1	0000
r2	0000
r3	0000
r4	0000
r5	0000
r6	0000
r7	0000
r8	0000
r9	0000
rA	0000
rB	0000
rC	0000
rD	0000
rE	0000
SP [rF]	0800
IP	0000

Status register

N	Z	IF2	IF1
☐	☐	✓	✓

Figure 5 The Registers window: (a) in the basic simulated processor; (b) in the extended simulated processor

The default contents of all locations in data memory are 0000. The user can return the contents of all locations to this default value by selecting Reset from the Options menu and then selecting Data memory.

The user can alter the contents of any location in data memory by double-clicking in the Contents field alongside the address of that location and then entering the new contents and pressing Enter.

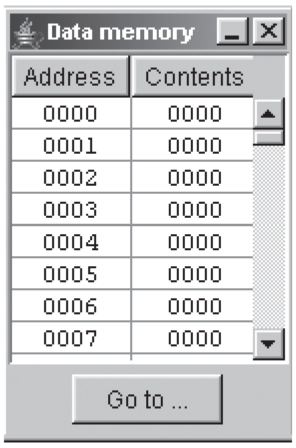
The contents can be edited using the cut, copy and paste facilities in the Edit menu.

Blocks of data can be deleted by first highlighting them in the Contents field and then cutting. The contents of these locations become 0000.

Blocks of data can be moved by first highlighting them in the Contents field, then cutting, then clicking in the Contents field alongside the first address which is to hold the data block, then pasting.

Data can also be copied and pasted elsewhere in program memory.

Clicking on the Go to button at the bottom of the window opens a dialogue box which prompts the user to enter an address in data memory; the Data memory window then scrolls to show that address.



Address	Contents
0000	0000
0001	0000
0002	0000
0003	0000
0004	0000
0005	0000
0006	0000
0007	0000

Figure 6 The Data memory window

4.5 The menus

The File menu

Open

This allows the user to open an existing file, which will contain a program and associated data memory contents and register contents.

The default sub-folder the first time the Open command is chosen is the CT programs sub-folder. This sub-folder will have been created as part of the software installation process and resides within the hierarchy of folders My Documents\T224\Simulated processor\ unless stipulated otherwise by the user during the installation process. CT programs remains the default sub-folder for the Open command until the user opens a program from another sub-folder.

When the new program and data are loaded, any previous instructions, data values and register contents which the user may have been working with are lost. Hence the user is prompted, where necessary, to save any work before the new program is loaded.

Merge

This brings in a program from a specified file and appends it to the program already in the program memory and shown in the Program editor window. If the new program and the existing program have instructions in the same locations(s) in program memory, the instructions in the new program will overwrite the instructions in the existing program in those locations.

Merge also brings in any associated data and appends it to the data already in the data memory. If the new program and the existing program have associated data in the same locations(s) in data memory, the data associated with the new program will overwrite the data associated with the existing program in those locations.

The registers are not affected by the Merge command.

If appropriate, the user is prompted to save any work before the new program is merged.

Save

If a program was retrieved from the CT programs sub-folder, it is necessary to use the Save As option to save it to another folder (thus preventing programs supplied by the course team from being overwritten).

Otherwise, Save saves the program and associated data memory contents and register contents with the same file name as the file from which they were retrieved, thus over-writing the previously saved version of the program, data memory and register contents.

Save As

This opens a dialogue box so that the user can enter the new file name and/or choose a new folder. The default sub-folder the first time the command is chosen is My programs, which will be in My Documents\T224\Simulated processor\ unless stipulated otherwise by the user during the installation process. Save As then saves the program and associated data memory contents and register contents.

Save As does not allow files to be saved to the CT programs sub-folder.

Save workspace

This allows the user to save the current size and window arrangement of the simulated processor screen. These then become the default.

Print

This allows the user to print the contents either of the currently highlighted window or of all windows. Except in the case of the registers, it does not print out any details for locations whose contents are zero. When 'print all' is selected, the contents of the Program memory window are not printed, as this would duplicate what is shown in the contents of the Program editor window.

Exit

This closes the simulation program. If appropriate, it prompts the user to save the current program first.

The Edit menu

Cut

This allows highlighted instructions in the Program editor or highlighted contents in the Data memory to be cut.

Copy

This allows highlighted instructions in the Program editor or highlighted contents in the Data memory to be copied.

Paste

This allows cut or copied instructions from the Program editor to be inserted where the user clicks. In the case of a block of instructions, the block will be inserted such that the first instruction goes where the user clicks.

Paste also allows cut or copied contents in the Data memory to be inserted where the user clicks. In the case of a block of data, the block will be inserted such that the first data item goes where the user clicks.

The Options menu

Reset

This allows the user to reset any of: Data memory; Program memory; Registers. In the extended version of the simulation it also allows the user to reset the Interrupt system. All of these can also be reset simultaneously. The user is asked to confirm the reset before it is implemented.

Reset returns the chosen entity to the state it has when the simulation is first opened.

Set trace speed

This allows the user to select from Fast, Medium or Slow for the trace speed. The default is Fast.

The View menu

This allows the user to open new Program editor and Data memory windows. If any of the four default windows has been closed, View allows the user to open it again.

The Devices menu

This menu is available only in the extended version of the simulation.

Input devices

This allows the user to open Scalepan, Switch bank or Interrupt button.

It is possible to have more than one copy of the windows for these devices open simultaneously.

Output devices

This allows the user to open LED bank, Analogue gauge, Simple 7-segment display, Smart 7-segment display or Buzzer.

It is possible to have more than one copy of the windows for these devices open simultaneously.

The Help menu

This takes the user to an electronic version of this Reference Manual via a hyperlinked contents page.

Appendix: ASCII codes

ASCII

Character	**Binary code**	**Hexadecimal code**	**Denary code**
null	000 0000	00	0
backspace	000 1000	08	8
horizontal tab	000 1001	09	9
line feed	000 1010	0A	10
carriage return	000 1101	0D	13
escape	001 1011	1B	27
space	010 0000	20	32
!	010 0001	21	33
"	010 0010	22	34
#	010 0011	23	35
$	010 0100	24	36
%	010 0101	25	37
&	010 0110	26	38
’	010 0111	27	39
(	010 1000	28	40
)	010 1001	29	41
*	010 1010	2A	42
+	010 1011	2B	43
,	010 1100	2C	44
-	010 1101	2D	45
.	010 1110	2E	46
/	010 1111	2F	47
0	011 0000	30	48
1	011 0001	31	49
2	011 0010	32	50
3	011 0011	33	51
4	011 0100	34	52
5	011 0101	35	53
6	011 0110	36	54
7	011 0111	37	55
8	011 1000	38	56
9	011 1001	39	57
:	011 1010	3A	58

;	011 1011	3B	59
<	011 1100	3C	60
=	011 1101	3D	61
>	011 1110	3E	62
?	011 1111	3F	63
@	100 0000	40	64
A	100 0001	41	65
B	100 0010	42	66
C	100 0011	43	67
D	100 0100	44	68
E	100 0101	45	69
F	100 0110	46	70
G	100 0111	47	71
H	100 1000	48	72
I	100 1001	49	73
J	100 1010	4A	74
K	100 1011	4B	75
L	100 1100	4C	76
M	100 1101	4D	77
N	100 1110	4E	78
O	100 1111	4F	79
P	101 0000	50	80
Q	101 0001	51	81
R	101 0010	52	82
S	101 0011	53	83
T	101 0100	54	84
U	101 0101	55	85
V	101 0110	56	86
W	101 0111	57	87
X	101 1000	58	88
Y	101 1001	59	89
Z	101 1010	5A	90
[	101 1011	5B	91
\	101 1100	5C	92
]	101 1101	5D	93
^	101 1110	5E	94
_	101 1111	5F	95
‘	110 0000	60	96

a	110 0001	61	97
b	110 0010	62	98
c	110 0011	63	99
d	110 0100	64	100
e	110 0101	65	101
f	110 0110	66	102
g	110 0111	67	103
h	110 1000	68	104
i	110 1001	69	105
j	110 1010	6A	106
k	110 1011	6B	107
l	110 1100	6C	108
m	110 1101	6D	109
n	110 1110	6E	110
o	110 1111	6F	111
p	111 0000	70	112
q	111 0001	71	113
r	111 0010	72	114
s	111 0011	73	115
t	111 0100	74	116
u	111 0101	75	117
v	111 0110	76	118
w	111 0111	77	119
x	111 1000	78	120
y	111 1001	79	121
z	111 1010	7A	122
{	111 1011	7B	123
¦	111 1100	7C	124
}	111 1101	7D	125
~	111 1110	7E	126
delete	111 1111	7F	127

Course team list

Academic staff

Bernie Clark, *Production Course Chair*

Geoff Einon

David Gorham, *Presentation Course Chair*

Reza Latif-Shabgahi

Mike Meade

Tony Nixon

Adrian Poulton

Richard Seaton

Mirabelle Walker

Production staff

Deirdre Bethune, *Course Secretary*

Colin Bluck, *Project Officer*

Philippa Broadbent, *Buyer, Materials Procurement*

Roger Courthold, *Graphic Artist*

Sarah Crompton, *Graphic Designer*

Daphne Cross, *Assistant Buyer, Materials Procurement*

Tony Duggan, *Learning Projects Manager*

Joanne Fellows, *QA Engineer*

Alison George, *Project Manager*

David Gosnell, *Software Designer*

Roger Harris, *Production Course Manager*

Lori Johnston, *Editor*

Karen Lemmon, *Compositor*

Deborah Mairs, *Presentation Course Manager*

Jane Moore, *Editor*

Jon Owen, *Graphic Artist*

Val Price, *Rights Executive*

Colin Thomas, *Software Designer*