Computer Architecture and Organization

8051 Project 2: memset()

1 Project's goal

The goal is to develop the assembly version of the memset() C function. The project will be carried out in several steps starting with simple building blocks and combining them later into more complex ones.

2 About memset()

2.1 Standard definition

memset() is a standard C function.

NAME
memset - fill memory with a constant byte

SYNOPSIS

```
#include <string.h>
void *memset(void *s, int c, size_t n);
```

DESCRIPTION
The `memset()` function fills the first n bytes of the memory area pointed to by `s` with the constant byte `c`.

RETURN VALUE
The `memset()` function returns a pointer to the memory area s.

2.2 Project implementation

Contrary to the standard function definition, the function implemented in this project should have no return value. Also, in order to adapt the code to the 8051 architecture we specify the function parameters in a more restrictive way:

```
void memset(unsigned char *s, unsigned char c, unsigned int n);
```

Hint: Develop your code in such a way that if $n == 0$ no data is written to RAM.

3 Implementing memset()

Project name: memset

3.1 memset() for 8051's internal RAM

Write an assembly program which writes the byte c into n consecutive memory locations in the internal RAM data area starting from address s. Assume that:

- c is stored in R5,
- s is stored in R7,
- n is stored in R3.

3.2 memset() for 8051's external RAM, $n < 256$

Modify your program from Section 3.1 such that:

- c is written into external RAM,
- s is 16-bit wide and is stored in R6/R7 (most significant byte (MSB) in R6).

Hints:

- Use `MOVX` to `MOV` data into external RAM.
- Use the 16-bit `DPTR` register to write data to 16-bit addresses (`DPTR` contains the address that should be operated on).
- Note the special instructions (`MOVX` and `INC`) operating on `DPTR` (refer to the 8051 instruction list).
- To set the value of `DPTR` use the registers `DPH` and `DPL`, which are the MSB and the LSB of `DPTR`, respectively.
- Using `EQU` redefine the constant `START` (see provided project files) to be 16 bit long.
- Use the `LOW` and `HIGH` assembler operators to extract the LSB and MSB, respectively, from 16-bit constants defined with `EQU`.
- Enter `x:1234h` in the Memory Window to see the contents of the external RAM starting from address 1234h.

3.3 memset() for 8051's external RAM, n 16-bit wide

Modify your program from Section 3.2 such that it operates correctly when n is 16-bit wide and stored in R2/R3 (MSB in R2).

Hint: Make sure that your code works correctly for all the combinations of parameters as specified in Section 6.

3.4 Packaging

Put your program from Section 3.3 into a subroutine called `MEMSET_SUB`.

4 Passing function parameters to subroutines

Project name: memset-8051

4.1 Preparation

Unlike in the 80x86 world, where function parameters are usually passed via stack, for the 8051 it is common to pass the function arguments either via reserved memory areas or via registers. In this project we use passing by registers. For background information, refer to the online help of Keil's μVision (You can start here: Select *Help* → *Open books window*, then choose *Complete User's Guide* from *Tools User's Guide* collection. Open then *Cx51 Compiler's User Guide*, go to *Advanced Programming* → *Interfacing C to Assembler* and from *Function Parameters* choose *Passing in Registers*).

Follow the standard described there while developing the macro `MEMSET` in the next section. See Keil μVision on-line help, topic „Defining standard macros“ for more information on usage and limitations of standard macros.

4.2 MEMSET macro

1. Prepare an assembler macro named `MEMSET`, which will:

   - take three arguments: `S`, `CH`, `N`, where:
     `S` corresponds to *s* of the memset() function
     `CH` corresponds to *c* of the memset() function
     `N` corresponds to *n* of the memset() function
   - set the register values R2, R3, R5, R6 and R7 from the macro parameters
   - `CALL` the unmodified subroutine `MEMSET_SUB` from Section 3.4.

   Hints:

   - Use the `LOW` and `HIGH` assembler operators to extract the LSB and MSB, respectively, from `S` and `N`.
   - You can assume that *s* is passed as a 2-byte pointer, not a generic pointer.

2. Write an assembler program which uses the macro to set 0246h bytes of external RAM starting at address 1234h to 55h. This should be equivalent to the following C code:

```
memset (0x1234, 0x55, 0x0246);
while (1);
```

5 Using assembly subroutines as C-functions

Project name: memset-from-c

5.1 Introduction

In order to be able to use assembly coded subroutines from C programs several conventions must be observed. They are described briefly below. The reference is the Keil online documentation. You may find also this tutorial helpful.

5.1.1 Parameter passing

This topic was discussed in Section 4. The code already adheres to Keil guidelines.

5.1.2 Function/subroutine naming

When calling a function `func` from C-code and passing its parameters via registers, the Keil tools expect to find a label with the name `_func` or `_FUNC` (the case does not matter) in one of the assembly modules.

5.1.3 Exporting of routines

In order for a label name to be found it must be exported from the assembly module. For the C-called function `func` it is achieved by the following assembler statement:

```
PUBLIC _func
```

5.1.4 Putting the code into a relocatable memory segment

The Keil tools will combine the assembly and C-code together. This includes placing the code at an appropriate memory address. The memory allocation is performed for whole blocks of memory called segments.

In order to avoid memory space conflicts, where different code pieces are placed at the same memory location, the assembly code must be put into a *relocatable code segment*.

Refer to the file `cao_memset.a51` provided with the project assignment for an explanation how this is done.

5.1.5 Name clashes

In order to avoid name clashes with the standard `memset` function, our assembly `memset` implementation shall be called as `cao_memset()`.

5.2 Calling assembly code from C

5.2.1 C code

Complete the C program in the `main.c` file.

5.2.2 Assembly subroutine

1. Read the code provided in the `cao_memset.a51` file and understand how code segments are defined.
2. Copy your subroutine `MEMSET_SUB` from Section 3.4 in the `cao_memset.a51` file.
3. Change the label such that it matches the Keil conventions and can be found by the tools.
4. Export the label by making it public.

5.2.3 Complete program

1. Build the project. Make sure there are no errors and no warnings.
2. Run the program. Check that the correct memory locations are set as expected.
3. Analyze the code inserted at location 0000.
4. Analyze the code inserted in place of the C code. Compare it to the code written in Section 4.2.

6 Final test

Ensure that your assembly implementation called from C is REALLY working. Test the following cases:

```
cao_memset(0x1234, 0x55, 0x0246);
cao_memset(0x1234, 0x66, 0x0200);
cao_memset(0x1234, 0x77, 0x0101);
cao_memset(0x1234, 0x88, 0xff01);
cao_memset(0x1234, 0x99, 0x0000);
```