Práctica 1: Desarrollo de código para el procesador ARM

Profesores: Javier Resano, Darío Suárez, Enrique Torres, María Villarroya

{jresano, dario, enrique.torres, mvg}@unizar.es

Proyecto Hardware
Área de Arquitectura y Tecnología de Computadores
Departamento de Informática e Ingeniería de Sistemas

Objetivos

- Optimizar el rendimiento de un juego acelerando las funciones computacionalmente más costosas.
  - entender funcionalidad del código suministrado
  - realizar versión en ensamblador ARM optimizado
  - medir rendimiento respecto
    al código optimizado generado por el compilador
  - documentar los resultados.
- Gestionar el tiempo de trabajo del proyecto correctamente en función de la disponibilidad de acceso al sistema específico.

Objetivos

- Conocer la estructura segmentada del procesador ARM 7
- Interactuar con una placa real y ser capaces de ejecutar en ella
  - Gestionar la entrada/salida, desarrollar en C las rutinas de tratamiento de interrupción
- Familiarizarse con la generación cruzada de código ARM y su depuración
  - siguiendo el contenido de los registros internos del procesador y de la memoria
- Desarrollar código en ensamblador y combinarlo con código en C
  - **ARM**: adecuado para optimizar el rendimiento
  - Entender la finalidad y el funcionamiento de las ***Aplication Binary Interface***, ABI (**AATPCS**)
- Analizar y medir rendimiento
- Optimizar código (a nivel de ensamblador) y Optimizaciones del compilador
- Redactar memorias técnicas

Bibliografía y Material disponible 1/2

- Básica:
  - David Seal; **ARM Architecture Reference Manual**; 2ª Ed, Addison-Wesley, 2001 (versión antigua disponible en la web de la asignatura)
- Complementaria:
  - A. Sloss, D. Symes, C. Wright; **ARM system Developer's Guide**; Morgan Kaufman, 2004
  - Steve Furber; **ARM System-on-chip Architecture**; 2ª Ed, Addison-Wesley, 2000
  - William Hohl; **ARM Assembly Language. Fundamentals and Tecniques**; CRC Press, 2009
- Enlaces
  - www.arm.com/documentation
  - www.arm.com/gnu

Bibliografía y Material disponible 2/2

- Breve resumen del repertorio de instrucciones ARM
- Documentación original del micro
- Códigos fuente iniciales y *Linker script*
- Directrices sobre cómo redactar la memoria

- Ejemplos y documentación arm-Keil
  - https://www.keil.com/dd/chip/4354.htm
- Foros y Wiki en Moodle

Entorno de trabajo

- Utilizaremos como entorno de trabajo Keil µVision con una cadena de herramientas (*toolchain*) para compilación cruzada:
  - LPC2105 simulado por el µVision Debugger

Estructura de la práctica

- Parte A (1 semana):
  - Objetivos:
    - Familiarizarse con el entorno y los repertorios de instrucciones
    - Crear proyectos, compilar y enlazar
    - Depurar sobre la placa y sobre el simulador.
    - Ejecutar paso a paso en alto nivel y en ensamblador el código suministrado
      - Observar registros y memoria
      - Breakpoints
      - …
    - Analizar rendimiento y medir tiempos
  - Preparar de cara a la siguiente sesión:
    - Programar en ensamblador **ARM optimizado** la función más costosa en computo**.**

Estructura de la práctica

- Parte B (2 semanas):
  - Reprogramar en ensamblador **ARM optimizado** la función más costosa en computo (previo)**.**
  - Observar código generado por compilador optimizador en -O0, -O1, -O2, …
  - Comparar las opciones (original, ARM y C optimizado) teniendo en cuenta:
    - Tamaño del código
    - Nº de instrucciones ejecutadas
    - Tiempo de ejecución.
  - Debéis optimizar lo máximo posible los códigos en ensamblador, pero respetando la misma estructura que el código C y la modularidad.
  - **Verificar automáticamente el funcionamiento** (equivalencia funcional)

Evaluación de la práctica

- Fechas estimadas:
  - Entrega de esta parte: **14-15** de Octubre
  - Entrega de la memoria: **21** de Octubre
  - Las fechas definitivas se publicarán en la página web de la asignatura (moodle)

Realización de la memoria

- Resumen ejecutivo (**una cara máximo**).
- Análisis de rendimiento (*profiling*)
- Descripción de las optimizaciones realizadas al código ensamblador
- Resultados de la comparación entre distintas versiones de la función
  - Descripción de los resultados
- Descripción de los problemas encontrados en la realización de la práctica y sus soluciones
- Firma y autoría.
- Código fuente del apartado B comentado. Además, la **función ARM** debe incluir una **cabecera** en la que se explique cómo funciona, qué **parámetros** recibe, dónde los recibe y qué **uso** se da a cada **registro** (ej. en el registro 4 se guarda el puntero a la primera matriz)

Esquema

- **Descripción de la práctica**
- Sistema de memoria
- Fuentes y ayudas de C

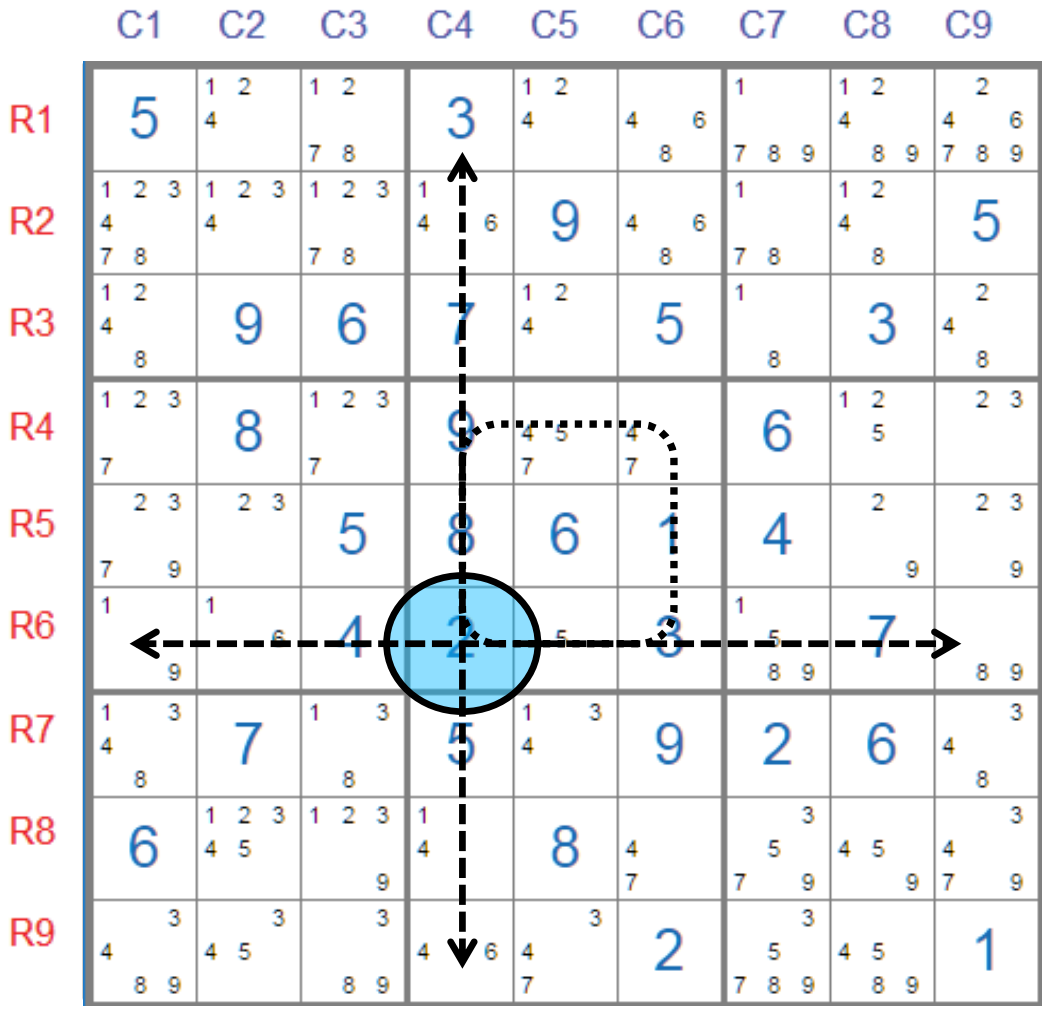
¿Sabéis hacer sudokus?

	C1	C2	C3	C4	C5	C6	C7	C8	C9
R1	5			3					
R2					9				5
R3		9	6	7		5		3	
R4		8		9			6		
R5			5	8	6	1	4		
R6			4			3		7	
R7		7		5		9	2	6	
R8	6				8				
R9						2			1

El cálculo de los candidatos es la clave

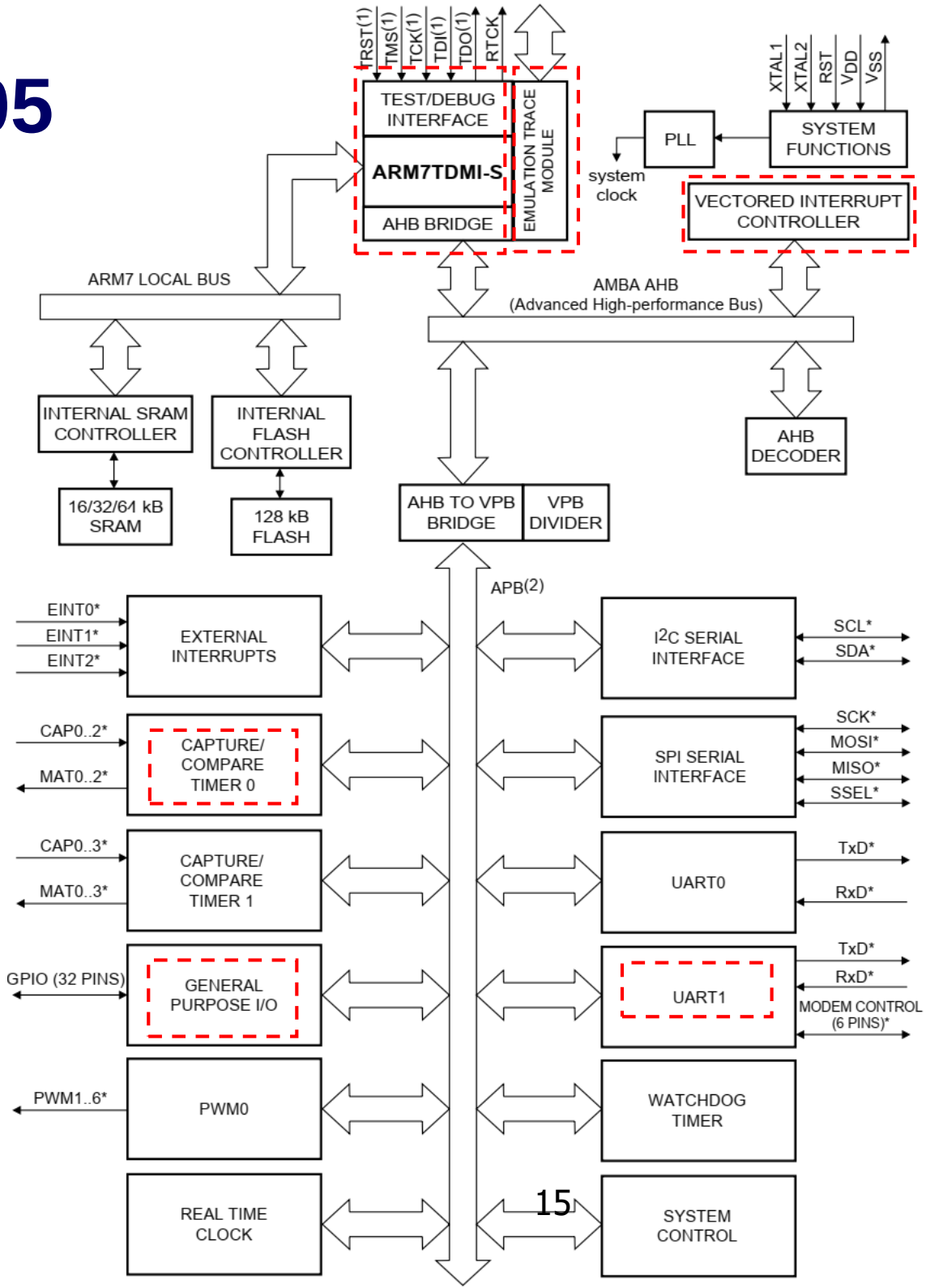
	C1	C2	C3	C4	C5	C6	C7	C8	C9
R1	5	1 2 4	1 2 7 8	3	1 2 4	4 6 8	1 7 8 9	1 2 4 8 9	2 4 6 7 8 9
R2	1 2 3 4 7 8	1 2 3 4	1 2 3 7 8	1 2 4 6	9	4 6 8	1 7 8	1 2 4 8	5
R3	1 2 4 8	9	6	7	1 2 4	5	1 8	3	2 4 8
R4	1 2 3 7	8	1 2 3 7	9	2 4 5 7	4 7	6	1 2 5	2 3
R5	2 3 7 9	2 3	5	8	6	1	4	2 9	2 3 9
R6	1 2 9	1 2 6	4	2	2 5	3	1 5 8 9	7	2 8 9
R7	1 3 4 8	7	1 3 8	5	1 3 4	9	2	6	3 4 8
R8	6	1 2 3 4 5	1 2 3 9	1 4	8	4 7	3 5 7 9	4 5 9	3 4 7 9
R9	3 4 8 9	3 4 5	3 8 9	4 6	3 4 7	2	3 5 7 8 9	4 5 8 9	1

El cálculo de los candidatos es la clave



Chip principal: LPC2105

- System-on-Chip (Soc):
  - Procesador ARM7TDMI
  - Controladores E/S
    - Temporizadores
    - Controlador de interrupciones
  - Controlador INT. Vectorizadas
  - EmbeddedICE
  - Depuración puerto JTAG
  - Y muchos más componentes
    - GPIO
    - UART
    - Timers



¿Qué tenemos que aprender?

- Cómo configurar el entorno:

- Gestión del hardware del sistema utilizando C:
  - Entender las estructuras que genera el compilador a partir del código fuente (especialmente la pila de programa) [**2**]

Cuadricula - Celda

- 9 x 9
- Celda 16bits
  - 4 bits Valor (1..9)
  - 1 bit Pista
  - 1 bit Error
  - 9 bits vector candidatos

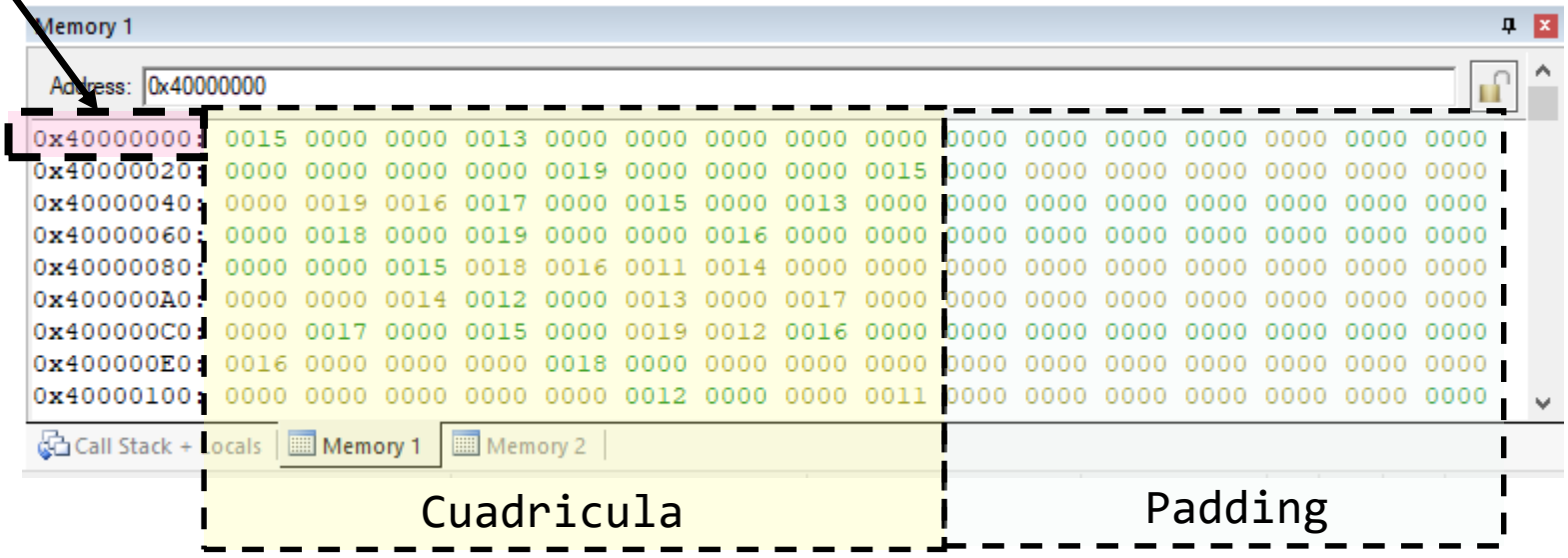
	C1	C2	C3	C4	C5	C6	C7	C8	C9
R1	5			3					
R2					9				5
R3		9	6	7		5		3	
R4		8		9			6		
R5			5	8	6	1	4		
R6			4			3		7	
R7		7		5		9	2	6	
R8	6				8				
R9						2			1

Candidatos									E	P	Valor			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

¿Cómo veremos el sudoku en el simulador?

- La cuadrícula del sudoku está almacenado en memoria
- hay que adaptar el tamaño de la ventana y ponerlo en formato "`unsigned short`" para que se visualice bien
- Cada posición son dos bytes que incluyen el número y la información adicional de los candidatos. Incluye padding

Posición de comienzo de la cuadricula (dependerá del compilador)



Esquema

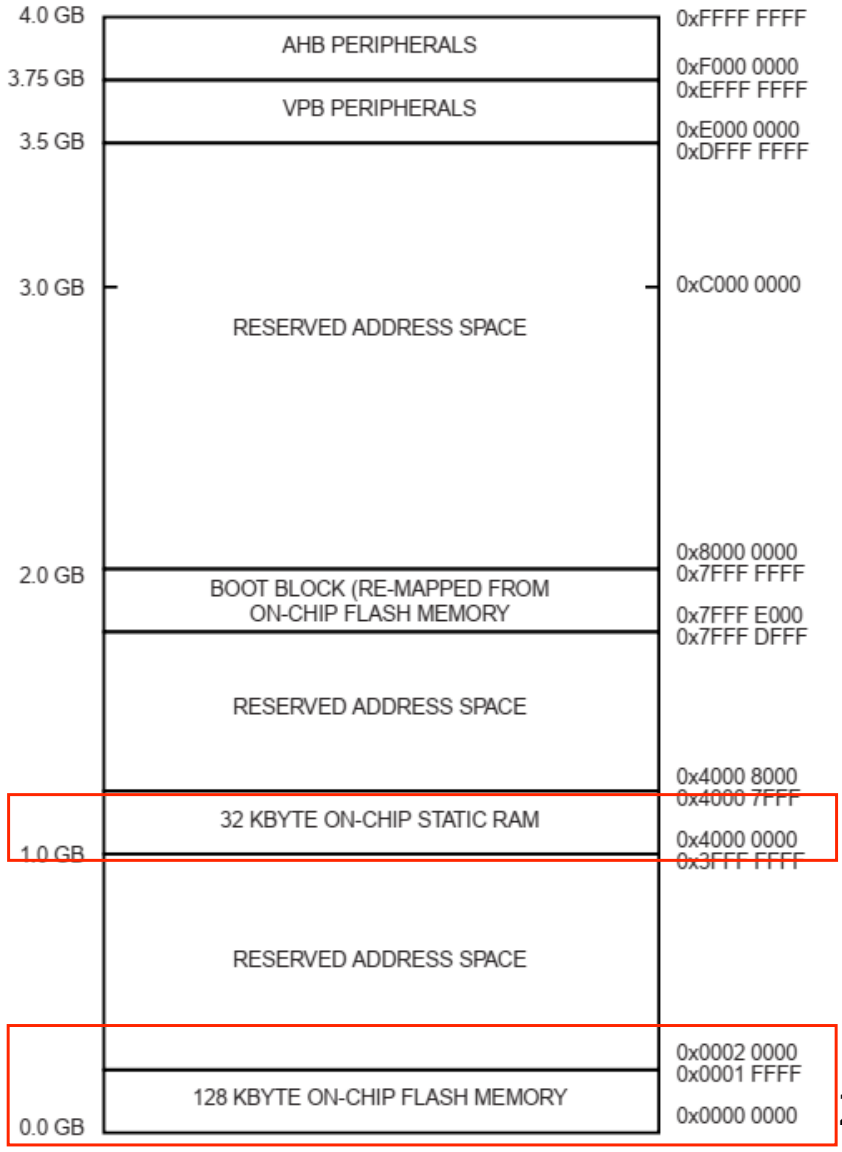
- Descripción de la práctica
- **Sistema de memoria**
- Fuentes y ayudas de C

Espacio de direcciones ARM7TDMI

- Principales características:
  - Direcciones de 32 bits (4GB direccionables)
  - Espacio compartido por E/S y memoria
    - E/S localizada en memoria: los puertos de E/S forman parte del espacio de direcciones
    - La distinción es externa al procesador

Espacio de direcciones del LPC2105

- Espacio dividido zonas
  - 128 kB Flash memory system
    - Código y datos
  - 32 kB RAM estática
    - Accesible como 8, 16 o 32 bits
  - Vector Interrupciones en flash o RAM
  - Registros E/S



Espacio de direcciones del LPC2105

- Zona de registros especiales
  - Ocupa el último tramo [0xE000_0000-0xFFF_FFFF]
  - La entrada/salida se configura escribiendo/leyendo en estos registros

Variables en C

- visibilidad, alcance y cualificadores
  - global, local
  - `const (#define, enum)`
  - `volatile`
  - `extern`
  - `static`
    - en globales
    - en locales
    - en funciones

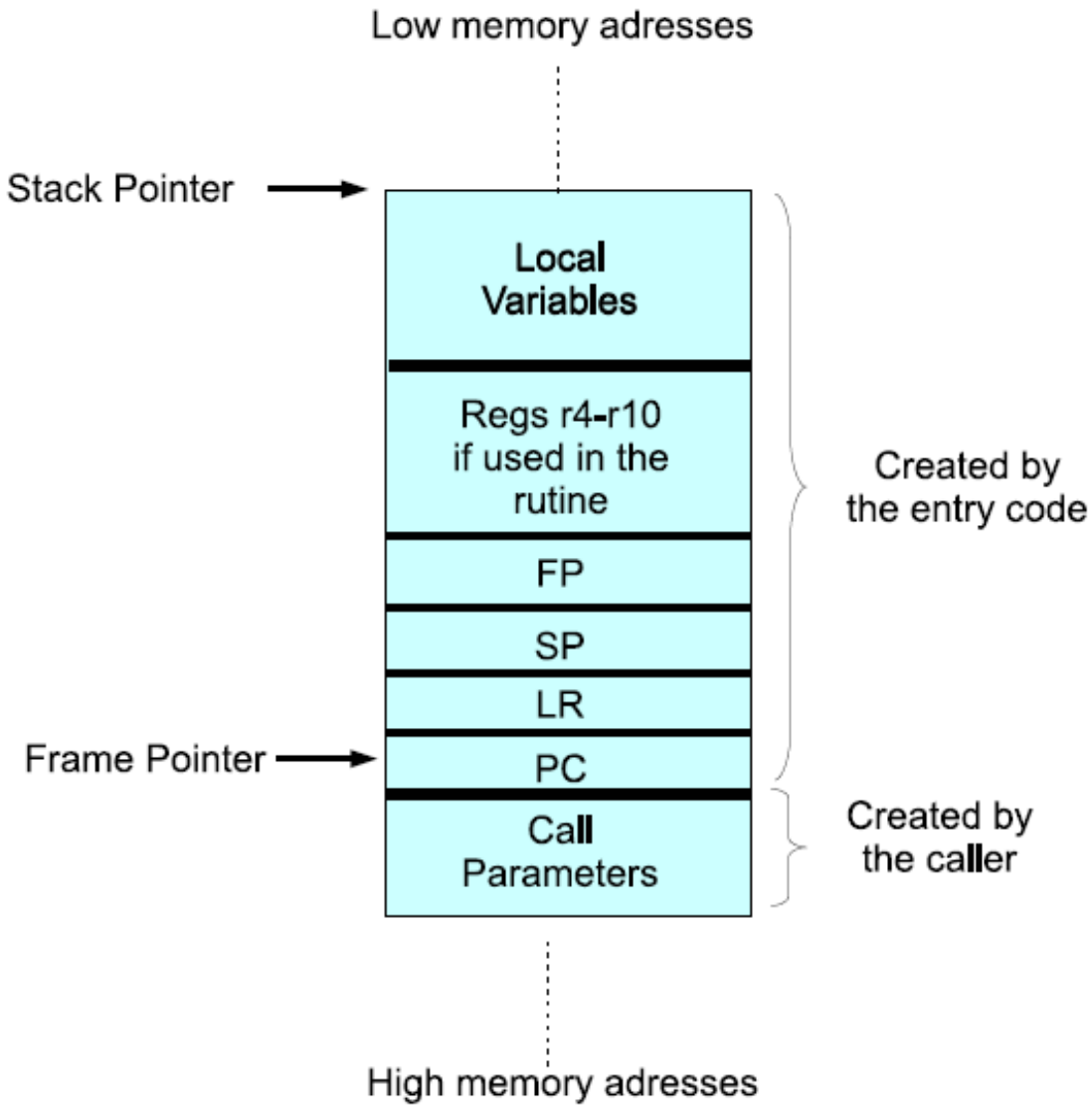
Recordatorio: Marco de pila

- Estructura de una rutina:
  - Prólogo

```
MOV     IP, SP
STMDB   SP!, {r4-r10,FP,IP,LR,PC}
SUB     FP, IP, #4
SUB     SP, #SpaceForLocalVaribles
```

  - Epílogo:

```
LDMDB   FP, {r4-r10,FP,SP,PC}
```



Esquema

- Descripción de la práctica
- Sistema de memoria de la placa S3CEV40
- **Fuentes y ayudas de C**

Ficheros de partida

- **`Startup.s:`** inicialización y tabla de vectores (inst. de salto)
- **`LPC210x.H :`** macros para usar nombres simbólicos
- `sudoku_2021.c`
  - Funciones principales

```
int main (void) {

  …
  // el empotrado nunca termina

}
```

Startup.s

- Fichero en ensamblador
- Se ejecuta tras CPU reset
- Inicializa CPU
  - Pilas de los diferentes modos
  - Vector Interrupciones

stdint.h - inttypes.h

- Un char en C:
  - ¿Qué tamaño y rango tiene?
  - ¿Tiene signo?
- Un int en C:
  - ¿Qué tamaño y rango tiene?

```
 /* exact-width signed integer types */
typedef   signed          char int8_t;
typedef   signed short     int int16_t;
typedef   signed           int int32_t;
    /* exact-width unsigned integer types */
typedef unsigned          char uint8_t;
typedef unsigned short     int uint16_t;
typedef unsigned           int uint32_t;
```

LPC210x.H

- Fichero cabecera del fabricante
  Macros para usar nombres simbólicos
  - Controlador de Interrupciones vectorizadas
  - GPIO
  - Periféricos

```
#ifndef __LPC210x_H
#define __LPC210x_H

/* Vectored Interrupt Controller (VIC) */
#define VICIRQStatus   (*((volatile unsigned long *) 0xFFFFF000))
#define VICFIQStatus   (*((volatile unsigned long *) 0xFFFFF004))
…
/* General Purpose Input/Output (GPIO) */
#define IOPIN          (*((volatile unsigned long *) 0xE0028000))
#define IOSET          (*((volatile unsigned long *) 0xE0028004))
```

celda.h

```
#ifndef CELDA_H
#define CELDA_H

#include <inttypes.h>

/* Cada celda se codifica en 16 bits
 * bits [15,7]: los 9 bits más significativos representan el vector de candidatos,
 * si el bit 7 + valor - 1 está a 0, valor es candidato, 1 en caso contrario
 * bit 6: no empleado
 * bit 5: error
 * bit 4: pista
 * bits [3,0]: valor de la celda */

enum { BIT_CANDIDATOS = 7 };

typedef uint16_t CELDA;

/* elimina el candidato del valor almacenado en la celda indicada */
static inline void
celda_eliminar_candidato(CELDA *celdaptr, uint8_t valor) {
    *celdaptr = *celdaptr | (0x0001 << (BIT_CANDIDATOS + valor - 1));
}
```

Sudoku_2021.c

```
…
// MAIN
int main (void) {
    #include "tableros.h"
    int correcto = sudoku9x9(cuadricula_C_C, cuadricula_C_ARM, cuadricula_ARM_ARM,
                                cuadricula_ARM_C, solucion);
}
```

Sudoku_2021.h

```
/* Tama□os de la cuadricula */
/* Se utilizan 16 columnas para facilitar la visualizaci□n */
enum {NUM_FILAS = 9,
      PADDING = 7,
      NUM_COLUMNAS = NUM_FILAS + PADDING};
…
/* declaracion de funciones visibles en el exterior */
int sudoku9x9(CELDA cuadricula_C_C[NUM_FILAS][NUM_COLUMNAS],
        CELDA cuadricula_C_ARM[NUM_FILAS][NUM_COLUMNAS],
        CELDA cuadricula_ARM_ARM[NUM_FILAS][NUM_COLUMNAS],
        CELDA cuadricula_ARM_C[NUM_FILAS][NUM_COLUMNAS],
        CELDA solucion[NUM_FILAS][NUM_COLUMNAS]);

void
candidatos_propagar_c(CELDA cuadricula[NUM_FILAS][NUM_COLUMNAS],
                      uint8_t fila, uint8_t columna);

/* declaracion de funciones ARM a implementar */
int
candidatos_actualizar_arm(CELDA cuadricula[NUM_FILAS][NUM_COLUMNAS]);
```

tableros.h

```
…
#include "celda.h"

static CELDA
cuadricula_C_C[NUM_FILAS][NUM_COLUMNAS] =
{
0x0015, 0x0000, 0x0000, 0x0013, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0, 0, 0, 0, 0, 0, 0,
0x0000, 0x0000, 0x0000, 0x0000, 0x0019, 0x0000, 0x0000, 0x0000, 0x0015, 0, 0, 0, 0, 0, 0, 0,
0x0000, 0x0019, 0x0016, 0x0017, 0x0000, 0x0015, 0x0000, 0x0013, 0x0000, 0, 0, 0, 0, 0, 0, 0,
0x0000, 0x0018, 0x0000, 0x0019, 0x0000, 0x0000, 0x0016, 0x0000, 0x0000, 0, 0, 0, 0, 0, 0, 0,
0x0000, 0x0000, 0x0015, 0x0018, 0x0016, 0x0011, 0x0014, 0x0000, 0x0000, 0, 0, 0, 0, 0, 0, 0,
0x0000, 0x0000, 0x0014, 0x0012, 0x0000, 0x0013, 0x0000, 0x0017, 0x0000, 0, 0, 0, 0, 0, 0, 0,
0x0000, 0x0017, 0x0000, 0x0015, 0x0000, 0x0019, 0x0012, 0x0016, 0x0000, 0, 0, 0, 0, 0, 0, 0,
0x0016, 0x0000, 0x0000, 0x0000, 0x0018, 0x0000, 0x0000, 0x0000, 0x0000, 0, 0, 0, 0, 0, 0, 0,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0012, 0x0000, 0x0000, 0x0011, 0, 0, 0, 0, 0, 0, 0
};
…
static CELDA
solucion[NUM_FILAS][NUM_COLUMNAS] = {
0x9a15, 0xfa00, 0x9e00, 0xeb13, 0xfa00, 0xab80, 0x1f00, 0x3a00, 0x0a80, 0, 0, 0, 0, 0, 0, 0,
0x9800, 0xf800, 0x9c00, 0xeb00, 0xfa19, 0xab80, 0x9f00, 0xba00, 0x8a95, 0, 0, 0, 0, 0, 0, 0,
0xba00, 0xfa19, 0xbe16, 0xfb17, 0xfa00, 0xbb95, 0xbf00, 0xba13, 0xba80, 0, 0, 0, 0, 0, 0, 0,
0xdc00, 0xfc18, 0xdc00, 0xfb99, 0xd380, 0xdb80, 0xf516, 0xf600, 0xfc80, 0, 0, 0, 0, 0, 0, 0,
```

Sudoku_2021.c

```
…
/* ************************************************************************
 * calcula todas las listas de candidatos (9x9)
 * necesario tras borrar o cambiar un valor (listas corrompidas)
 * retorna el numero de celdas vacias */

/* Init del sudoku en codigo C invocando a propagar en C
 * Recibe la cuadricula como primer parametro
 * y devuelve en celdas_vacias el n?mero de celdas vacias
 */
static int candidatos_actualizar_c(CELDA cuadricula[NUM_FILAS][NUM_COLUMNAS])
{
    int celdas_vacias = 0;
    uint8_t i;
    uint8_t j;

        //borrar todos los candidatos
        ...
        //recalcular candidatos de las celdas vacias calculando cuantas hay vacias
        ...
        //retornar el numero de celdas vacias
        ...
}
```

Sudoku_2021.c

```
void candidatos_propagar_c(CELDA cuadricula[NUM_FILAS][NUM_COLUMNAS],
        uint8_t fila, uint8_t columna) {
    uint8_t j, i , init_i, init_j, end_i, end_j;
    /* puede ayudar esta "look up table" a mejorar el rendimiento */
    const uint8_t init_region[NUM_FILAS] = {0, 0, 0, 3, 3, 3, 6, 6, 6};

    uint8_t valor = celda_leer_valor(cuadricula[fila][columna]); /* valor que se propaga */

    for (j=0;j<NUM_FILAS;j++) /* recorrer fila descartando valor de listas candidatos */
        celda_eliminar_candidato(&cuadricula[fila][j],valor);

    for (i=0;i<NUM_FILAS;i++) /* recorrer columna descartando valor de listas candidatos */
        celda_eliminar_candidato(&cuadricula[i][columna],valor);

    /* determinar fronteras región */
    init_i = init_region[fila];
    init_j = init_region[columna];
    end_i = init_i + 3;
    end_j = init_j + 3;

    for (i=init_i; i<end_i; i++) {/* recorrer region descartando valor de listas candidatos *
        for(j=init_j; j<end_j; j++) {
            eliminar_candidato(&cuadricula[i][j],valor);
```