به نام خدا

پروژه پایانی درس معماری کامپیوتر

عنوان پروژه: پردازش تصویر به زبان اسمبلی

نام استاد: جناب آقای دکتر راعی

نام گردآورنده: مارال مرداد

شماره دانشجویی: 9723148

پاییز 99

توضیح کد:

کد دارای دو بخش کلی می باشد. بخش اول پروژه که شامل کرنل های گوسی هموار ساز و تشخیص لبه می باشد و بخش دوم آن که مقدار دهی تنظیمات میکروکنترلر است.

در بخش اول دو آرایه دو بعدی به نام های Image و ImageN تعریف شده است(مانند شکل1) که ردیف های آن ها به صورت dcd (32 بیت) و ستون های آن ها به صورت dcb (8 بیت) تعریف شده این ماتریس ها در اصل ماتریس 17*17 هستند که حاوی اطلاعات عکس اصلی padding شده و عکس نویزی padding شده می باشند. در انتهای کد هم یک ماتریس 15*15 به نام NewImage به همین شکل تعریف شده با این تفاوت که به صورت readwrite تعریف شده است. این ماتریس برای ذخیره سازی عکس بعد از اعمال فیلتر ها می باشد.

```
Image   dcd     row0,row1,row2,row3,row4,row5,row6,row7,row8,row9,row10,row11,row12,row13,row14,row15,row16
row0    dcb     129,129,109,153,143,118,158,144,42,102,175,157,133,114,177,72,72
row1    dcb     129,129,109,153,143,118,158,144,42,102,175,157,133,114,177,72,72
row2    dcb     102,102,110,157,109,97,111,114,6,102,99,86,122,122,183,151,151
row3    dcb     83,83,107,103,133,137,39,130,2,103,110,75,93,94,135,121,121
row4    dcb     105,105,99,144,81,116,80,125,48,102,107,108,77,95,100,108,108
row5    dcb     95,95,100,66,85,108,66,126,22,71,53,98,88,147,137,100,100
row6    dcb     192,192,73,79,119,119,136,113,7,112,85,80,141,132,36,87,87
row7    dcb     144,144,144,135,122,172,122,118,0,137,101,140,85,102,127,118,118
row8    dcb     32,32,28,27,0,25,0,29,42,38,14,0,34,0,0,59,59
row9    dcb     114,114,130,100,184,113,124,97,8,104,151,58,62,65,120,140,140
row10   dcb     122,122,44,116,78,82,141,93,0,111,57,63,99,61,110,139,139
row11   dcb     116,116,107,169,45,159,106,123,0,112,121,97,116,133,101,102,102
row12   dcb     68,68,40,158,88,100,143,115,57,141,153,114,48,62,117,81,81
row13   dcb     137,137,69,78,117,106,85,126,19,91,87,82,100,82,83,112,112
row14   dcb     145,145,144,132,95,121,148,85,67,72,166,153,87,80,77,127,127
row15   dcb     131,131,141,166,134,171,129,128,9,112,116,74,113,73,64,122,122
row16   dcb     131,131,141,166,134,171,129,128,9,112,116,74,113,73,64,122,122
```

شکل1

در کد از 5 تابع به نام های GetIndexN ,GetIndex، SetIndex، GaussianKernel و EdgeDetectKer استفاده شده است. در ابتدای برنامه هم یک فلگ IsGaussionKernel تعریف شده است که در صورت 1 بودن آن فیلتر گوسی و در صورت 0 بودن آن فیلتر تشخیص لبه روی عکس اعمال می شود.

تابع GetIndex برای دسترسی به هر درایه از ماتریس Image می باشد به این صورت که ردیف و ستون را به عنوان ورودی دریافت میکند (چون ردیف ها به صورت dcd تعریف شده ردیف را در 4 ضرب میکند)

و در خروجی محتوای درایه مورد نظر را در رجیستر r2 می دهد.(شکل2)

تابع GetIndexN هم دقیقا عملکردی مشابه تابع GetIndex دارد با این تفاوت که امکان دسترسی به هر درایه از ماتریس ImageN را فراهم می کند.

```
GetIndex    MOV32   r2, #4
            MUL     r0, r0,r2
            ;MUL    r1, r1,r2
            MOV32   r2, #Image
            LDR     r2, [r2,r0]
            LDRB    r2, [r2,r1]
            MOV     pc, lr
```

شکل2

تابع SetIndex برای مقدار دهی در ماتریس نهاییِ NewImage تعریف شده است. در ورودی r0، r1 و r2 را دریافت میکند که به ترتیب ردیف و ستون و مقداری که باید در آن خانه از ماتریس قرار داده شود هستند.(شکل3)

```
SetIndex    MOV32   r6, #4
            MUL     r0, r0,r6
            ;MUL    r1, r1,r6
            MOV32   r6, #NewImage
            LDR     r6, [r6,r0]
            STRB    r2, [r6,r1]
            MOV     pc, lr
```

شکل3

تابع GaussianKernel برای اعمال فیلتر گوسی به عکس نویزی می باشد. باید توجه کرد که چون در این تابع توابع GetIndexN و SetIndex صدا زده می شوند و تعریف تابع باید به گونه ای باشد که بعد از اجرای آن، دستوری که بعد از دستور صدا زدن این تابع قرار دارد، اجرا شود (یعنی pc مقدار قبلی خود را بگیرد.) در ابتدای این تابع باید مقدار link register را در یک رجیستر ذخیره کرد که در انتهای تابع مقدار رجیستر در pc ریخته شود.

در این تابع رجیستر های r3 و r4 وردی می باشند که به ترتیب سطر و ستون درایه مورد نظر برای اعمال فیلتر گوسی را مشخص می کنند. این تابع درایه مورد نظر را درایه وسط یک ماتریس 3*3 در نظر گرفته و مقادیر آن را نظیر به نظیر در ماتریس کرنل گوسی ضرب می کند و همه ی این مقادیر را جمع کرده، بر 16 تقسیم میکند و در درایه ای از ماتریس NewImage می ریزد که که هر دو عدد سطر و ستون آن یک واحد از r3 وr4 کمتر باشند.

EdgeDetectKernel هم مانند GaussianKernel یک تابع با ورودی های r3 و r4 می باشد که درایه ورودی را درایه مرکزی یک ماتریس سه در سه در نظر می گیرد و درایه های ماتریس تشخیص لبه را نظیر به نظیر در آن ضرب می کند. برای ضرب کردن عدد منهای چهار در درایه وسط میتوان با استفاده از 2 تا

شیفت چپ آن را در 4 ضرب کرد و بعد با دستور SBCS مقدار منفی آن را با مقدار رجیستری که حاوی جمع مقادیر Index های قبلی می باشد جمع کرد.

برای اینکه هریک از این کرنل های گوسی و تشخیص لبه روی تمام پیکسل های عکس اعمال شود برای هر یک از آن ها یک for تو در تو در نظر گرفته شده است. r8 و r9 به عنوان شمارنده های این for های تو درتو از پیکسل 1 تا 15 روی Image حرکت می کنند و به این ترتیب کرنل روی تمام پیکسل های عکس اعمال می شود.(شکل4و5)

```
GaussionStart
            MOV32   r8, #1
            MOV32   r9, #1
MyLoop2     MOV     r3, r8
            MOV     r4, r9
            BL      GaussianKernel
            ADD     r9, r9, #1
            CMP     r9, #16
            BNE     MyLoop2
            ADD     r8, r8, #1
            MOV32   r9,#1
            CMP     r8, #16
            BNE     MyLoop2
```

شکل4

```
            MOV32   r8, #1
            MOV32   r9, #1
MyLoop1     MOV     r3, r8
            MOV     r4, r9
            BL      EdgeDetectKernel
            ADD     r9, r9, #1
            CMP     r9, #16
            BNE     MyLoop1
            ADD     r8, r8, #1
            MOV32   r9,#1
            CMP     r8, #16
            BNE     MyLoop1
            B       loop    ;skip gaussion filter
```

شکل5

در بخش دومِ کد با توجه به موارد خواسته شده رجیستر های مورد نظر را تعریف شده اند و از صفحه ی 50 دیتا شیت آدرس شروع آن ها برداشته شده و به آن آدرسی مقدار offset (که آن هم برای هر رجستر در دیتا شیت ذکر شده) مربوطه اضافه شده است.(شکل 6)

```
RCC_AHB1ENR     EQU     0x40023830
GPIOA_MODER     EQU     0x40020000
GPIOB_MODER     EQU     0x40020400
GPIOB_OTYPER    EQU     0x40020404
GPIOA_OSPEEDR   EQU     0x40020008
GPIOB_OSPEEDR   EQU     0x40020408
GPIOA_PUPDR     EQU     0x4002000C
GPIOB_PUPDR     EQU     0x4002040C
GPIOA_IDR       EQU     0x40020010
GPIOB_ODR       EQU     0x40020414
```

شکل6

رجیستر RCC_AHB1ENR برای فعال سازی کلاک پورت های A و B، رجیستر های GPIOA_MODEو GPIOB_MODER برای تعیین مد وردی و خروجی pinA0 وpinB1 ،رجیستر GPIOB_OTYPER برای تعیین pinB1 از نوع push pull ، رجیستر های GPIOA_OSPEEDRو GPIOB_OSPEEDR برای تعیین سرعت پین های مذکور، رجیستر های GPIOA_PUPDR و GPIOB_PUPDR هم برای تعیین pull up/down می باشند.

برای مقدار دهی به این رجیستر ها باید دقت کرد که این رجیستر ها 32 بیتی هستند و ما فقط یک بیت یا دو بیت آن ها را (بسته به نوع تعریف آن ها در دیتا شیت که به هر پین یک بیت اختصاص داده شده یا دو پین) میخواهیم مقدار مشخصی در آن بریزیم و نمی خواهیم مقادیر بقیه بیت ها تغییری کند به همین دلیل همان طور که در توضیحات پروژه آمده از AND و OR استفاده می کنیم.

در ابتدای کد به صورت شکل 7 با 1 شدن pinA0 برنامه شروع به کار میکند و در انتهای برنامه بعد از آن که عکس نهایی آماده شد و ماتریس NewImage پر شد، مانند شکل 8 pinB1 را 1 می کند.

```
                ;decide to start or not
ReadPin         MOV32   r10, #GPIOA_IDR
                LDR     r11, [r10]
                ANDS    r11, r11, #1
                BEQ     ReadPin
```

شکل7

```
;turn on output ready flag
MOV32   r10, #GPIOB_ODR
LDR     r11, [r10]
ORR     r11, r11, #2     ;0(
STR     r11, [r10]
```

شکل 8

نتایج نهایی :

Memory 1

Address: 0x20000000

0x20000000:	119	123	137	133	126	135	113	076	098	138	141	128	135	141	113
0x2000000F:	105	116	129	125	113	111	092	063	084	111	110	112	128	143	135
0x2000001E:	096	107	118	119	106	095	082	064	081	100	093	094	109	125	128
0x2000002D:	098	103	106	105	099	094	087	070	078	095	093	093	104	113	111
0x2000003C:	114	099	090	096	101	102	089	065	069	083	091	105	115	108	099
0x2000004B:	141	110	097	110	121	119	089	060	076	092	098	113	113	098	094
0x2000005A:	120	104	094	099	107	101	074	056	074	086	085	086	081	081	089
0x20000069:	081	079	075	077	076	068	055	048	062	067	055	049	049	064	085
0x20000078:	092	086	091	098	093	089	067	049	071	080	061	055	061	085	111
0x20000087:	109	101	107	106	110	116	083	053	079	093	082	084	091	107	125
0x20000096:	097	101	111	101	109	121	091	065	091	109	098	094	099	104	106
0x200000A5:	088	091	107	106	109	118	098	076	101	120	102	085	088	096	096
0x200000B4:	111	098	101	107	109	112	095	073	090	113	105	087	081	091	103
0x200000C3:	135	128	122	119	124	122	095	068	086	117	115	096	082	087	110
0x200000D2:	136	143	145	141	143	135	097	066	089	113	105	095	080	083	109
0x200000E1:	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000
0x200000F0:	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000

نتیجه ی حاصل پس از اعمال فیلتر گوسی(نتیجه نهایی ماتریس 15*15 سبز رنگ می باشد.)

Memory 1

Address: 0x20000000

```
0x20000000: 058 060 066 067 055 066 033 125 030 061 059 061 067 063 073
0x2000000F: 063 085 034 053 093 058 036 102 039 087 061 066 065 038 058
0x2000001E: 065 072 078 069 026 089 027 124 031 056 073 060 054 081 061
0x2000002D: 062 059 041 075 090 062 024 124 029 055 062 089 044 078 063
0x2000003C: 070 053 085 057 066 069 029 127 020 089 069 066 079 040 060
0x2000004B: 049 096 093 054 065 055 041 123 021 072 088 045 066 097 071
0x2000005A: 032 025 020 034 027 029 007 129 010 029 031 038 027 024 029
0x20000069: 122 109 123 114 129 125 128 033 128 119 112 105 111 116 114
0x20000078: 035 031 025 030 017 034 009 127 002 022 051 049 034 057 029
0x20000087: 054 087 080 056 097 070 022 126 028 076 080 071 055 054 063
0x20000096: 059 051 066 071 051 089 025 119 037 066 054 057 075 056 070
0x200000A5: 055 084 035 098 039 050 041 105 043 058 060 054 080 057 068
0x200000B4: 058 069 094 037 091 077 031 101 043 058 060 079 067 073 054
0x200000C3: 061 075 060 060 071 053 038 118 042 085 044 065 051 088 054
0x200000D2: 061 058 062 050 062 066 032 125 027 066 076 050 081 056 058
0x200000E1: 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
```

نتیجه ی حاصل پس از اعمال فیلتر تشخیص لبه