

어셈블리프로그래밍설계 및 실습

Term project 결과보고서

설계제목: matrix convolution

설계일자: 2021년 11월 22~11월 29일

제출일자: 2021년 11월 30일 (화요일)

학 과: 컴퓨터정보공학부

담당교수: 이준환 교수님

실습분반: 화0,1 목2

학 번: 2018202060

성 명: 이 준 형

1. 제목 및 일정

A. 제목

matrix convolution

B. 일정

기간	1주차							2주차
항목								
11월	23	24	25	26	27	28	29	30
제안서 제출								
문제정의 및 정보수집								
coding								
coding 검증								
Report 작성								
최종 제출								

2. Project specification

이번에 설계한 프로젝트의 전체적인 흐름은 매트릭스의 값을 불러와서 패딩을 시켜서 저장한 후, 그 매트릭스의 행렬곱을 시키면서 전에 했던 SUM과 점점 합쳐가면서 9번 연산이 완료되면 그 곱이 짝수행 짝수열인지 확인을 하고 짝수행 짝수열이면 저장을 시켰습니다. 곱하면서 sub_sampling을 진행했습니다.

PADDING과정

START

주어진 매트릭스와 패딩이 되는 매트릭스를 활용하기 위해 이 함수에서 로드를 시키고 카운팅을 할 레지스터를 미리 초기화했습니다.

PADDING

먼저 맨 첫째줄의 패딩하는 과정인데 맨 첫째줄에 로드를 하고 SAVE를 2번씩 시키는 과정입니다.

PADDING_LOOP

OFFSET을 4씩 증가시키며 행렬의 값을 61번 STR시켰습니다.

CHECK_PADDING_LOOP

한줄의 행렬의 패딩이 끝났는지 확인을 하는 함수입니다.

INNER2_PADDING_LOOP

두번째줄부터 맨 마지막줄 바로 위에줄의 패딩하는 반복문 함수입니다.

STOP

맨 아랫줄이 되기전까지 패딩이 되었으면 OFFSET을 -256시켜 맨 아랫줄 패딩을 시작합니다.

PADDING_LOOP3

맨 아랫줄 패딩하는 과정인데, 맨 첫째줄 패딩하는 과정이랑 흡사합니다.

PADDING_END

패딩이 끝난경우 convolution에 대비해서 미리 작은 음수의 offset을 이용해서 패딩 매트릭스의 첫번째 원소를 가르키게 했습니다.

MATRIX CONVOLUTION 과정

DOMUL

먼저 booth multiplication을 위해서 sign과 exponent, mantissa를 분리 시키는 과정입니다. 그 후 R2에 32의 절반인 16이 아닌 28의 절반인 14를 주어서 RADIX_4연산을 시켰습니다.

RADIX_4

R2가 0이 될때까지 아래의 연산 과정을 진행시켰습니다.

SHIFT, ADD_1A, ADD_2A, SUB_1A, SUB_2A

MUL_NOMALIZATION

RADIX_4연산이 끝난 후, 2^{28} 인 134217728과 비교를 해서 exponent를 더해주었습니다.

MUL_ADDING

sign과 exponent mantissa가 전부 합치는 과정입니다.

MUL_FINISH

곱한것과 전에 곱해서 합쳐진것을 더하는 과정으로 넘어가는 함수입니다.

DOADD

먼저 입력받은 값들이 0인지 아닌지 확인을 합니다. 그 후에 mul을 하기위해서 sign, exponent, mantissa를 추출했습니다.

ADDING_LOOP, ADDING_MANTISSA, ADDING_ADDING

2^{24} 인 16777216을 비교한 후, mantissa를 더할지 안더할지 결정 후, 더해줍니다,

SUB_SAMPLING 과정

CHECK_STATE

앞에서 곱하고 더한과정의 3번째인지 6번째인지 9번째 인지 확인을 하고, 그 번째에 맞는 함수를 call합니다.

THIRD_SIXTH_CAL

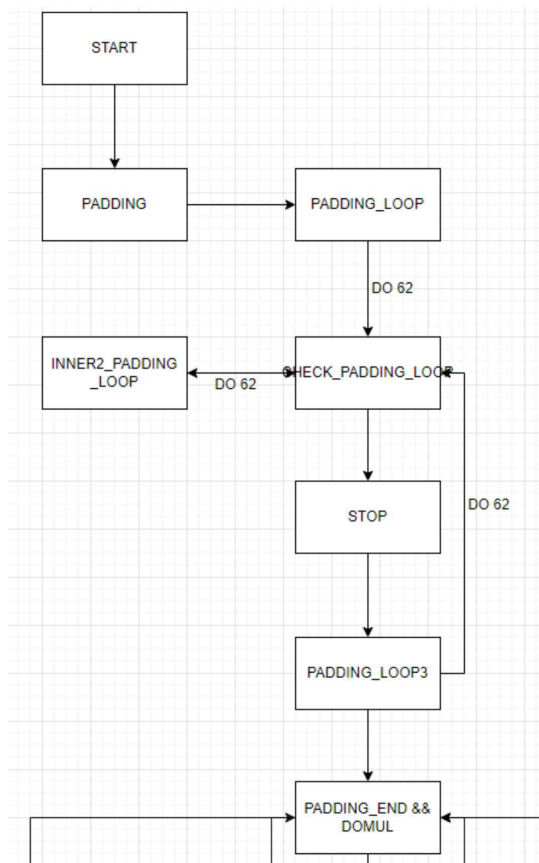
3번째나 6번째인 경우 offset을 252만큼 증가시켜서 다시 연산을 진행하게 했습니다.

LAST_CAL

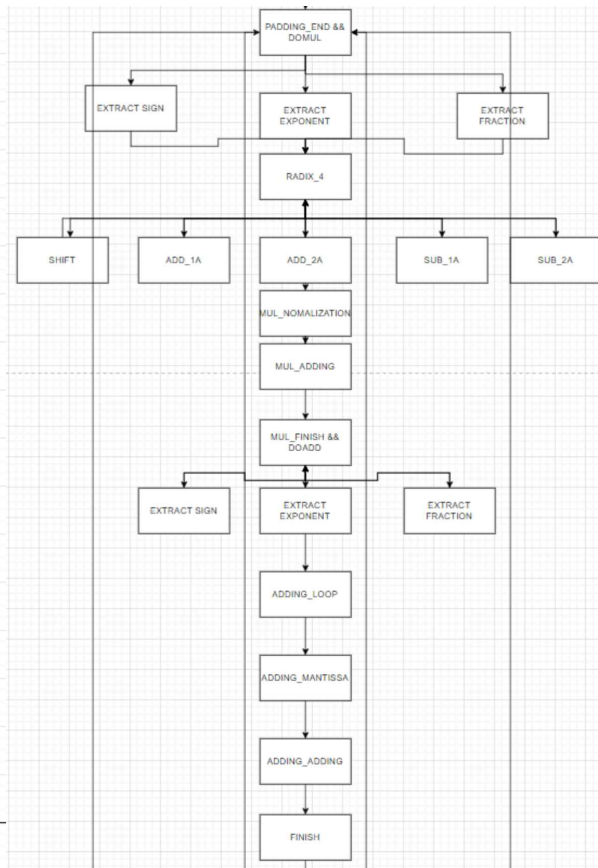
9번째 연산인 경우 sub_sampling을 바로 진행하기위에서 자기 위에위에인 원소를 가르키기 위해서 offset을 -532를 시켜주었습니다.

아래는 각 함수의 flowchart입니다.

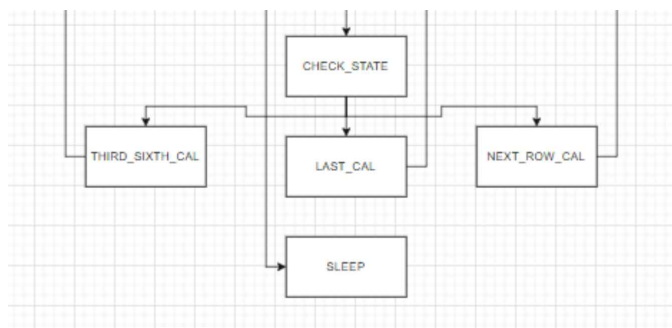
패딩하는 과정입니다.



matrix_convolution과정입니다.



sub_sampling하는 과정입니다.



함수들의 전체 flowchart입니다.



3. Algorithm

padding

패딩이 완료될 66*66의 매트릭스를 생각할때는 그 줄의 맨 끝들이 한번씩 더 호출이 되는것을 확인할 수 있습니다. 그래서 한번씩 더 호출했고, offset을 -256, -252이런식으로 해주면서 값에 LDR하게 했습니다.

booth multiplication of booth algorithm

이번 프로젝트에서 사용한 booth multiplication은



이러한 부동소수점을 곱해주는 알고리즘입니다.

보통의 MUL이라는 명령어를 사용했을때는

$value = (-1)^s * (1.mantissa) * 2^{(exponent-127)}$ 을 이용해서 구하지만, MUL이라는 명령어를 없이 구현할 때 쓰는 알고리즘입니다.

수업시간에 배웠던 radix-2 방식으로 이용했을경우에는 32bits면 32번의 연산이 나와하지만 radix-4인 경우는 16번만 연산을하면 가능합니다. 하지만 경우의수가 2배가 더 많아집니다. radix-4의 경우는 총 경우의수가 8가지인데, 제가 코딩한 경우의수는 총 5가지입니다. 기존이랑 중복되는게 있어서 이와 같은 5가지를 이용했습니다. shift, add_2a, add_1a, sub_1a, sub_2a입니다.

이 알고리즘은 승수에 0을 더 추가하여 그 0부터 MSB가 될때까지 피승수와 의 상호작용을 통해서 곱해지는 알고리즘입니다. 저는 LSL, LSR을 통해서 왔다갔다하면서 Y와 X+1, X, X-1을 바꿔주고 shift나 결과에 2a나 a값을 빼주고 shift 해주는 과정을 14번을 진행한 후, 14번의 2배인 28을 2의 지수로 올려서 비교를 한 후, sign, exponent, fraction을 전부 더하는 식으로 했습니다.

곱해진 수와 기존에 더해졌던 수는 똑같이 sign, exponent, mantissa를 추출해서 exponent를 비교해서 더하는 식으로 했습니다.

위 과정을 진행하면서 1에 가까운 수를 더하는 경우가 있는데 그때 저는 cr을 0에 가까운 0x800000을 equ해서 define시켜주었습니다.

sub_sampling

제가 구현한 sub_sampling은 레지스터 두개를 카운팅 하게 해서 짝수행이 진행될때 한 레지스터를 진행시켜주고, 짝수열이 진행될때 한 레지스터를 진행시켜주었습니다. 그렇게 하면서 두개의 카운트가 모두 짝수일때 save를 시켜주었습니다.

4. Performance & result

A. Performance

Code size

```
Restricted Version with 32768 Byte
Currently used: 17240 Bytes (52%)
```

Code states

```
--Internal
| PC $      0x00000320
| Mode      Supervisor
| States     3129040
| Sec        0,00000000
```

조교님께서 말씀해주신 $\text{score}(\text{states} * \text{states} * \text{code size})$ 는

$1.68794966384384e+17$ 가 나옵니다.

이것은 168794963511869440입니다.

B. Result

Padding

처음에 float로 봄으로써 제대로 값을 받았는지 확인을 했습니다.

Memory 1												
Address [x10000000]												
0x10000000:	167	167	167	165	163	164	172	175	157	122	111	117
0x10000004:	119	122	130	136	140	141	141	141	143	144	144	144
0x10000008:	144	143	142	142	142	143	143	143	141	140	140	138
0x1000000C:	134	126	126	146	161	162	161	162	162	161	162	163
0x10000010:	203	205	150	124	129	131	131	133	134	134	135	139
0x10000014:	139	167	167	167	165	163	164	172	175	157	122	111
0x10000018:	119	119	122	130	136	140	141	141	141	143	144	144
0x1000001C:	144	144	143	142	142	142	143	143	143	141	140	140
0x10000020:	139	134	126	126	146	161	162	161	162	162	161	162
0x10000024:	172	203	205	150	124	129	131	131	133	134	134	135
0x10000028:	139	139	166	166	165	164	163	165	172	174	155	120
0x1000002C:	116	118	118	122	129	135	139	141	141	142	143	144
0x10000030:	143	143	143	142	142	141	141	143	143	142	141	140
0x10000034:	139	138	135	127	124	140	159	143	143	144	143	142
0x10000038:	142	167	184	208	164	164	157	128	130	131	133	135
0x1000003C:	120	108	108	165	165	165	164	168	172	171	152	120
0x10000040:	110	116	118	117	121	129	135	139	141	142	142	143
0x10000044:	144	143	143	143	142	142	141	141	143	143	142	141
0x10000048:	139	139	139	136	129	122	131	153	165	167	167	166
0x1000004C:	162	161	161	176	206	196	144	126	129	131	132	134
0x10000050:	130	98	69	69	165	165	165	166	167	171	171	168
0x10000054:	119	110	116	117	117	121	129	135	139	140	141	142
0x10000058:	143	143	143	143	142	140	140	140	139	141	142	141
0x1000005C:	139	139	139	139	137	131	123	127	148	148	148	167
0x10000060:	165	163	161	159	162	189	210	174	132	128	131	132
0x10000064:	130	98	67	61	61	166	166	167	169	171	172	169
0x10000068:	152	119	109	116	117	117	121	129	134	137	139	139
0x1000006C:	141	142	142	142	143	141	138	137	137	137	139	140
0x10000070:	140	139	139	138	139	137	132	125	127	145	161	164
0x10000074:	165	164	164	163	161	159	170	203	205	156	127	130
0x10000078:	127	96	67	62	63	63	168	168	168	171	171	169
0x1000007C:	167	152	118	108	115	116	116	121	129	133	136	138

처음에 입력받은 값이 제대로 save되었고 padding또한 제대로 된것을 확인했습니다.

그리고 22개씩 unsigned int로 볼때

```
2x10004180: 42C20000 42B60000 42B80000 42DC0000 42D60000 42A60000 428E0000 428E0000 427C0000 427C0000 42980000 430D0000 43360000 43180000 43250000 43370000 43270000 42CA0000 427C00
2x10004230: 428A0000 42900000 42980000 429C0000 42AE0000 42B80000 42A60000 42980000 42AC0000 42A00000 429A0000 428A0000 42B60000 42E80000 43010000 43080000 430C0000 430E0000 431000
2x10004280: 43150000 43170000 43180000 431B0000 431C0000 431D0000 431D0000 431D0000 431E0000 43210000 43260000 432D0000 43350000 433F0000 43490000 43500000 433E0000 42F20000 42DD00
2x100043D0: 42FC0000 43000000 42FC0000 42F80000 42DC0000 42C60000 42BA0000 42B60000 42B80000 42E80000 42B80000 42E80000 42940000 42940000 427C0000 427C0000 42980000 430D0000 433600
2x10004320: 43250000 43370000 43270000 42CA0000 427C0000 427C0000 42BA0000 42900000 42980000 429C0000 42AE0000 42B80000 42A60000 42980000 42BA0000 42AC0000 429A0000 42B60000 42B600
2x10004370: 43010000 43080000 430C0000 430E0000 43100000 43130000 43150000 43170000 43180000 431B0000 431C0000 431D0000 431D0000 431E0000 43210000 43260000 432D0000 43350000
2x100043C0: 43490000 43500000 433E0000 42F20000 42DD0000 42E80000 42FC0000 43000000 42FC0000 42F80000 42DC0000 42C60000 42BA0000 42B60000 42B80000 42E80000 42B80000 42940000 429400
2x10004410: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

66*66이 4356이니까 198줄이 나오는것을 확인할 수 있었습니다.

matrix convolution & sub_sampling

Memory 1											
Address: 0x60000000											
0x60000000:	121.319	120.129	121.251	122.011	73.8871	56.8061	59.6264	105.652	103.761	105.239	106.292
0x60000030:	106.713	106.292	105.66	106.142	105.944	104.668	103.972	92.031	75.4206	114.352	118.582
0x60000060:	119.444	129.363	118.303	94.2849	114.261	99.8186	101.01	103.24	120.477	120.279	122.313
0x60000090:	72.7109	56.4468	59.1192	105.231	103.34	105.312	106.069	106.652	106.354	105.66	105.014
0x600000C0:	105.721	104.47	103.824	105.081	81.0995	106.849	120.548	120.277	118.717	121.605	119.992
0x600000F0:	113.967	99.8207	114.091	72.8206	122.246	122.434	123.45	118.992	72.065	55.9622	58.7097
0x60000120:	102.423	104.172	105.226	105.871	105.821	104.324	103.815	104.113	104.754	104.111	103.813
0x60000150:	72.8231	99.851	119.568	119.224	119.224	117.415	127.564	107.339	100.145	99.6426	47.6796
0x60000180:	122.934	121.142	119.834	119.202	75.4327	58.9203	58.7097	105.229	102.075	103.899	105.089
0x600001B0:	106.279	109.564	115.505	117.363	113.196	105.696	103.128	101.02	72.4001	96.9422	115.691
0x600001E0:	117.327	115.059	116.077	119.844	111.319	62.8243	40.274	39.1968	124.04	55.775	109.107
0x60000210:	77.8076	60.5151	57.52	79.8946	100.737	103.403	110.957	101.916	104.47	110.871	121.678
0x60000240:	117.433	123.873	114.816	101.226	65.8122	97.0263	113.287	114.115	117.237	112.693	121.967
0x60000270:	100.146	38.6599	31.8612	37.2897	107.855	65.9273	102.088	120.055	76.8024	59.942	56.4624
0x600002A0:	107.534	97.0444	93.7568	114.581	103.119	108.132	118.582	131.187	122.096	125.883	118.805
0x600002D0:	57.3511	97.0128	114.306	80.5129	104.249	113.806	113.704	100.125	46.6389	31.3154	38.7928
0x60000300:	68.6368	58.1251	101.494	119.497	76.7522	60.1034	56.7376	78.483	107.58	73.3752	80.1777
0x60000330:	105.435	108.499	116.976	132.156	124.03	128.717	131.307	111.272	69.3213	95.2155	114.712
0x60000360:	71.1358	112.708	107.276	65.9372	49.3187	33.7114	61.1538	98.1953	65.3585	60.6542	100.616
0x60000390:	76.1948	59.0868	56.3032	78.3756	117.498	79.4742	97.359	102.528	105.944	109.971	124.019
0x600003C0:	123.336	126.57	128.817	130.645	114.041	110.658	114.487	66.1372	84.6208	125.776	100.521
0x600003F0:	52.2017	45.23	70.3939	114.555	60.086	59.2042	101.299	120.401	76.6314	59.6075	57.0085
0x60000420:	101.305	58.8645	105.949	102.166	107.16	118.81	128.41	132.019	118.922	122.715	124.585
0x60000450:	129.783	123.944	117.121	89.7706	116.382	130.343	108.309	39.0884	40.5235	61.9701	107.298
0x60000480:	54.9727	57.4973	103.048	122.58	78.5765	60.6494	56.7991	74.9199	114.679	75.0053	105.183
0x600004B0:	111.076	118.334	125.2	127.422	129.058	132.391	118.303	119.465	119.829	118.101	117.093
0x600004E0:	126.699	126.988	99.4101	33.8485	55.0938	102.575	118.453	118.173	53.9216	57.0058	103.944
0x60000510:	75.1473	60.3014	55.5457	75.7222	119.46	98.2901	103.462	106.217	110.002	114.017	117.818
0x60000540:	121.242	123.474	125.718	127.697	131.603	121.251	125.122	127.241	118.978	117.232	67.1505
0x60000570:	65.715	115.458	118.988	118.369	54.7768	70.8496	117.513	124.12	79.2939	55.819	55.4092
0x600005A0:	108.549	108.549	101.147	105.029	106.752	102.611	77.7367	91.8687	76.5109	75.8802	88.3358
0x600005D0:	121.234	123.496	122.859	113.724	112.42	91.6404	41.8329	61.0072	106.301	118.801	118.023
0x60000600:	57.5771	68.3987	71.741	124.122	79.3454	59.3847	55.4092	70.6935	106.706	111.917	101.21
0x60000630:	99.4818	66.9235	60.0995	60.0202	61.6443	67.5432	115.96	116.957	120.185	122.73	108.444
0x60000660:	97.4155	62.603	37.4311	66.9573	117.018	120.341	118.321	116.683	58.7669	63.0005	107.484
0x60000690:	75.4932	58.6795	55.4842	63.5919	111.118	115.82	100.217	95.9717	63.8034	59.1901	54.5551
0x600006C0:	65.1495	114.248	129.484	130.875	120.509	104.789	62.3285	92.6662	64.6528	56.8595	47.8463
0x600006F0:	120.409	121.135	119.497	77.463	57.7887	57.2513	101.799	124.766	80.1256	59.819	56.9469
0x60000720:	103.905	117.441	102.151	69.186	57.8916	57.5441	57.8406	82.9305	108.296	124.7	125.816
0x60000750:	127.175	105.976	57.0672	56.7895	50.5224	51.3958	62.4003	104.226	118.166	118.394	117.04
0x60000780:	56.7354	56.3211	101.303	125.1	80.8944	60.7244	57.7908	104.4	107.48	112.597	72.0464
0x600007B0:	52.5985	53.8698	67.7771	118.524	122.566	114.601	119.724	118.16	126.596	107.244	69.6027
0x600007E0:	52.5658	48.4193	75.4443	116.959	118.391	116.064	113.589	112.409	55.4685	54.9946	100.173
0x60000810:	81.3903	60.7859	57.4606	102.389	97.0821	60.8426	62.7544	53.8575	52.1933	49.8205	97.9459
0x60000840:	75.148	69.6782	81.095	125.741	121.629	71.7833	56.8023	58.4968	52.2481	57.5121	99.5785
0x60000870:	117.316	116.26	113.801	109.465	54.4881	54.911	102.077	126.534	75.4799	56.6277	58.2851
0x600008A0:	80.903	59.6023	57.1221	47.8578	46.5489	77.14	113.771	105.639	72.6346	99.2261	102.26
0x600008D0:	112.506	68.808	57.0238	61.0032	58.2274	68.1113	107.536	115.903	113.023	111.66	116.36
0x60000900:	57.7073	57.41	103.247	127.416	76.8396	63.2035	83.3513	63.7113	55.6945	56.4232	57.6449
0x60000930:	44.1876	100.712	111.447	122.082	122.094	114.515	122.074	118.294	55.8738	58.9524	61.8998
0x60000960:	55.1222	73.7954	114.643	114.229	112.182	111.03	123.598	120.08	65.8958	62.7839	100.033
0x60000990:	76.3573	62.1237	63.5591	66.7483	56.3762	42.4299	57.6203	51.5789	64.3859	71.439	67.9884
0x600009C0:	122.866	124.201	113.043	128.385	98.1493	57.542	57.7706	59.6208	57.6573	115.837	112.889
0x600009F0:	109.406	118.872	122.249	128.196	57.6912	63.1904	100.852	125.956	80.7625	62.3003	57.474
0x60000A20:	60.8673	55.2372	57.8609	66.5489	89.3791	66.4465	70.8992	109.472	117.391	119.16	109.968
0x60000A50:	122.695	74.3431	54.8083	52.8379	68.7725	105.946	114.577	111.97	110.989	114.286	128.953
0x60000A80:	62.8024	65.8191	105.509	126.367	89.1208	64.5954	64.6558	61.1897	62.71	56.0706	63.769
0x60000AB0:	64.189	59.8772	72.6703	107.541	113.704	113.393	114.393	122.002	110.053	62.3235	53.8348
0x60000AE0:	92.1583	113.922	113.323	110.705	113.962	121.893	131.98	133.319	54.4403	63.0457	106.83
0x60000B10:	103.096	60.2339	59.7319	59.2735	58.8495	68.9041	97.2313	57.4985	53.123	54.884	69.5471
0x60000B40:	110.257	108.128	110.324	118.841	89.9312	41.4542	55.9014	88.4866	97.4977	115.205	113.1
0x60000B70:	116.436	119.816	126.089	134.1	52.8347	59.4566	103.242	127.172	81.0691	58.1181	56.4939
0x60000BA0:	61.2818	69.617	83.2399	56.5035	58.9696	51.8698	56.8163	63.988	101.142	111.228	114.988
0x60000BD0:	67.8779	47.7071	59.1361	61.2119	99.5476	115.28	113.458	108.61	117.744	115.835	119.153
0x60000C00:	37.0251	100.908	126.033	97.1738	59.9311	53.5867	59.7251	56.4097	75.3643	64.6652	55.7166
0x60000C30:	51.4536	47.1708	53.2732	58.9302	72.9484	111.144	119.829	107.825	51.039	61.597	64.2442
0x60000C60:	94.9662	114.352	114.866	103.756	119.844	124.949	118.091	113.761	66.8878	66.8166	81.4627
0x60000C90:	59.0247	54.4444	55.4465	57.9769	54.1705	57.947	62.5988	59.5818	48.0223	50.8363	57.2349
0x60000CC0:	105.909	110.173	116.801	127.797	115.678	81.029	66.2854	68.044	98.6339	109.662	84.3193
0x60000CF0:	120.204	126.718	108.452	69.4294	66.3169	84.5169	78.505	126.242	100.72	46.5428	59.4394
0x60000D20:	58.4665	55.7072	74.0234	58.534	55.2239	53.7364	68.0415	68.1137	105.023	111.276	117.131
0x60000D50:	121.626	124.22	102.647	72.3463	106.973	107.899	97.9342	99.6882	116.075	107.9	63.2958
0x60000D80:	43.5294	81.0069	76.3724	126.872	104.421	37.2567	52.4061	58.5144	53.8762	50.5712	55.4515
0x60000DB0:	54.6361	52.3651	59.1574	70.2794	107.7	111.551	115.493	122.945	133.275	129.015	116.177
0x60000DE0:	108.377	107.618	103.769	123.225	127.555	112.169	55.4086	57.2509	40.501	82.6597	77.901
0x60000E10:	104.3	39.6814	49.2877	53.8922	41.4619	58.49	59.838	61.5887	58.2422	55.3637	79.2391
0x60000E40:	110.219	111.415	119.722	129.258	125.633	128.492	114.101	113.697	106.127	105.462	112.659
0x60000E70:	120.493	86.8779	60.9142	62.5678	40.8595	87.7599	84.7182	125.593	105.565	40.6618	38.1442
0x60000EA0:	41.1619	50.8779	63.2771	53.4076	42.5473	52.35	65.1042	100.315	112.043	112.12	112.197
0x60000ED0:	124.77	121.251	131.248	119.457	77.2544	37.5428	99.2976	125.951	118.053	67.8426	66.8341
0x60000F00:	63.6084	100.027	109.287	125.38	68.5652	73.5428	43.895	53.0124	40.442		

5. Conclusion

이번 프로젝트는 뿌듯함이 있었지만 아쉬움이 좀 더 많았던 프로젝트였던 것 같습니다. 먼저 padding을 하는 과정에서 3가지의 생각을 했습니다. 전체 패딩이 된 매트릭스를 저장할지, sub_sampling을 할 때 필요한 원소만 저장할 것인가, 아니면 처음부터 sub_sampling 할 때 필요한 padding만 저장할 것인가를 생각했습니다. 세번째 방법으로 진행하면서 size를 줄이려고 했는데 size를 줄이는 대신에 너무나 생각할게 많아서 시간이 부족한 저는 첫번째 과정인 가장 심플한 부분으로 진행하게 되어서 조금 아쉽습니다. 만약에 두번째 경우나 세번째 경우로 진행했을 경우, states수가 반이상이 줄어들 것 같습니다.

그 후 matrix_convolution과 sub_sampling을 같이 진행하기 위해서 짝수행과 짝수열을 카운팅 할 레지스터를 써야하는데, 레지스터가 모자랄때 sp를 이용해서 불렀다가 저장하라는 동기의 조언이 있었지만, 실습시간과 수업시간에 과제외에는 제대로 사용해본적이 없어서 r13(lr)과 r14(sp)를 그냥 카운팅하는 레지스터로 썼다는점도 매우 아쉽습니다. sub_sampling을 위해서 조교님께 여쭙본 stride라는 파라미터라는 제대로 사용해보고 싶었지만 조금 더 공부해야할 것 같습니다. multiplication이 100% 되지않아서 상당히 아쉬웠습니다. offset을 많이 건드려서 정확히 진행했다고 생각했는데 14번의 연산을 통해 진행을 하면서 덜 계산이 되었던 것 같습니다.

하지만 high level에서 느껴보지 못했던 감성을 느낄 수 있었던 좋은 과목과 프로젝트 였던 것 같습니다. 어떻게 high level과 다른지, 어떻게 assembly가 되는지도 배움으로써 highlevel언어로 구현하면서 assembly관점에서는 어떻게 진행될것인지 생각해보며 컴퓨터 전공에 한걸음 더 다가가는 것 같았습니다. 좋은 강의 해주셨던 조교님과 교수님들께 감사하다는 말씀 전하며 레포트 마치겠습니다.