Upcoming Assignments



Homework 1 due Monday Sept 22

Project 1s & 1m due Thursday Sept 18

This Lab Will Cover:



2. Memory and Struct Alignment



LC2K Instructions Compared to ARM





LC2K	ARM	Differences
add (R-type)	ADD	Also ADDI for constants, and SUB/SUBI for subtraction. (LC2K Subtraction: nor the 2nd with itself, add 1, add result to 1st one). MOVZ #0 zeros out reg. We use LSL, LSR for shifts. (Pseudo-instructions: MUL)
nor (R-type)	Load #-1 then EOR/SUB	Much easier to use ORR/ORRI, AND/ANDI, EOR/EORI. (Note: a NOT instruction exists in pseudocode for ARM overall, but not LEGv8)
lw (I-type)	LDURSW	This is for 32 bits. Also LDUR, LDURH, LDURB.
sw (I-type)	STURW	This is for 32 bits. Also STUR, STURH, STURB.
beq (I-type)	CMP then B.EQ	CBZ branch if zero (beq 0 regB offset), B for unconditional (beq 0 0 offset)
jalr (J-type)	BR (branch reg)	Using BL with #0 right before BR stores return address
halt (O-type)	End of file	
noop (O-type)	NOP	Technically not in LEGv8, but present in ARM overall
EECS 370		Fall 2025 3

ARM LEGv8 Reference Card



See the ARM LEGV8 reference card for a helpful overview of important LEGv8 instructions.

This can be found in the website under "References Material" in the "Course Resources" section.

Actification Acti	Semantics Semantics	Comments recited to recited to recited to the control of the cont
ADD XL XL XL XL XL XL XL	Sem: X 2 = 1 X 2 = 1 X 3 = 1 MXS: MIXS: MIXS:	resister to resister Page XXVC
ADDR	Sem (X2 =) (X2 =) (X3 =) (X4 =) (X4 =) (X5 =) (X6 =) (X6 =) (X7 =) (X8 =) (X8 =) (X1 =) (X2 =) (X3 =) (X4 =) (X	Comments Commen
A continued	Sem (X = 1) (X 2 = 1) (X 2 = 1) (X 2 = 1) (M MYS.) (M MYS.) (M MYS.)	10 FM AV. We see that the see t
Assembly Assembly code	Semi	Comments Commen
A continued by the co	Sems	In the Name of the
Assembly String	Sem: x2 = 1 x2 = 1 x2 = 1 x2 = 1 MIXS.	Comments Commen
See See See See See See See See See Se	Sem: (2 = 2) (2 = 2) (2 = 2) (3 = 2) (3 = 2) (3 = 2) (3 = 2) (4 = 2) (0 E1 De lumignod 4-055 O E1 De lumignod 4-055 O E1 De lumignod 4-055 COmments Anne 1 Anne
Transfer Operations NIBS XA, An, An, An, And And And Ansembly code	Sem: (2 = 1) (2 = 1) (2 = 1) (3 = 1) (4 = 1) (4 = 1) (5 = 1) (6 = 1) (7 = 1) (8 = 1) (8 = 1) (9 = 1) (9 = 1) (1 = 1	Comments Of the numbered 5 4000 Of the numbered 5 40
Transfer Operations	Sem: (2 = 1) (2 = 2) (2 = 3) (2 = 3) (3 = 4) (4 = 4) (Gommenti deader work and ino XI from Xn + s/mmg deader work lead ino XI from Xn + s/mmg word lead to bear 150 XI from Xn + s/mmg sign extent upper 23b 19 word lead to sever 160 XI from Xn + s/mmg mig. An outside upper 45b here lead to leave 160 XI from Xn + s/mmg mig. An outside upper 45b here lead to leave 160 XI from Xn + s/mmg mig. An outside upper 45b here lead to leave 160 XI from Xn + s/mmg mig. An outside the sign of the
Transfer Operations Assembly code	Semantics X2 = MIX6, 8181 X2 = MIX6, 8181 X3 = MIX6, 8181 X4 = MIX6, 8181 MX5, 8121 = X4 MX5, 8121 = X4 MX5, 8121 = X4 MX5, 8121 = X4	Comments doubte word testin into Trean Xin + Fairmin's age extend upper 32b word load to beer 32b Xi from Xin + Fairmin's age extend upper 32b word load to beer 32b Xi from Xin + Fairmin's zero extend upper 43b by the sold and the series (3b Xi from Xin + Fairmin's zero extend upper 45b pays in a series (2b Xi from Xin + Fairmin's zero extend upper 55b double would never 15b Xi from Xin + Fairmin's zero extend upper 55b word after from 15b Xin + Fairmin's zero extend upper 55b word after from 15b Xin + Fairmin's zero extend upper 55b word after from 15b Xin + Fairmin's zero extend upper 55b word after from 15b Xin + Fairmin's zero extend upper 55b word after from 15b Xin + Fairmin's zero extend from 15b Xin + Fairmin's zero extend from leafter 5c + 245b from 15b Xin - Control of the 15b Xin + Fairmin's zero extend from leafter 5c + 245b from 15b Xin - Control of the 15b Xin + Fairmin's zero extend from leafter 5c + 245b from 15b Xin - Control of the 15b Xin + Fairmin's zero extend from 15b Xin + Fairmin's zero extend from leafter 5c + 245b from 15b Xin - Control of the 15b Xin + Fairmin's zero extend from leafter 5c + 245b from 15b Xin - Control of the 15b Xin + Fairmin's zero extend from leafter 5c + 245b from 15b Xin - Control of the 15b Xin + Fairmin's zero extend from leafter 5c + 245b from 15b Xin - Control of the 15b Xin + Fairmin's zero extend from leafter 5c + 245b from 15b Xin - Control of the 15b Xin + Fairmin's zero extended from leafter 5c + 245b from 15b Xin - Control of the 15b Xin - Con
10 10 10 10 10 10 10 10	X2 = MIX6, #181 X2 = MIX6, #181 X2 = MIX6, #181 X2 = MIX6, #181 X3 = MIX6, #181 MIX #121 = X4 MIX #121 = X4 MIX #121 = X4	double work found in Nor from X ₁ = A feature of the double with death of the A feature of the catcal upon 3.25 were lated to found X ₂ + A feature of the catcal upon 4.25 were lated to found Y ₂ + A feature of the catcal upon 4.25 were lated to found the catcal the catcal upon 4.25 were lated for the C X ₂ + A feature of the catcal upon 5.25 were lated for the X ₂ X to X ₃ + A feature of the C X ₄ - A feature of the C X ₄
10 10 10 10 10 10 10 10	X2 = MIX6, #181 X2 = MIX6, #181 X2 = MIX6, #181 MIX3, #121 = X4 MIX3, #121 = X4 MIX3, #121 = X4 MIX3, #121 = X4	word load to be SC XN from Xn, chainer's sign except 13th is well so to love I XN from Xn + Animor's sign except a reset of the below to be the SN from Xn + Animor's sign except a reset of the below to the low SN from Xn + Animor's second upper 50s double west stee from Xn xN + Animor's second upper 50s double west stee from Xn xN + Animor's word stee from Xn xN + Animor's is weed stee from Xn + Animor's is weed stee from Xn + Animor's is weed stee from Xn + Animor's 24x 5 + Maricol member 24 + 24.53 24x 5 + Maricol member 24 + 24.53 24x 6 - Animor's second Xn = Dishart (N = 32) from N × 48.
Compared	X2 = M(X6, #18] X2 = M(X6, #18] X2 = M(X6, #18] M(X5, #12] = X4 M(X5, #12] = X4 M(X5, #12] = X4 M(X5, #12] = X4	is, word had to be very fifth Kimon No. 4 A faminely non-stand upon 48b the word had to be the SN Kimon XXxx + detailman zero, extend upon 58b double word state from XX to XX + defailman zero, extend upon 59b double word state from XX to XX + defailman or served state of the SN is XX + defailman or served state from lower 33b of XX to XX + defailman or served state from lower 15b of XX to XX + defailman or served state from lower 15b of XX to XX + defailman or served out XX and page 12b of XX + defail and the xX + defail and XX + defail
17.00 17.0	X2 = MJX6, #181 MIX5, #121 = X4 MIX5, #121 = X4 MIX5, #121 = X4 MIX5, #121 = X4	hely to take 1 to 8.5 X Y (m. Nr. 4, Ariman) are created upper 569 chacke versit store from X to X x + 4 starms) are created upper 569 chacke versit store from X to X x + 4 starms) are versit store from the versit 2 to X Y to X x + 4 starms? I went found from invert (ibo X to X x + 4 starms? I went found from invert (ibo X to X x + 4 starms? And x + 3 but started meanufact x - 4 x + 555 The x + 5 x +
STURN N. No.	MIX5, #12] = X4 MIX5, #12] = X4 MIX5, #12] = X4 MIX5, #12] = X4	diobels would are time NR No NA + fairmage word state from lower 33 or NR to NA + fairmage by word load from lower 15 or NR to NA + fairmage by two load from load 8 or NR to NA + fairmage 345 ≤ 9 fets injent for load from load 8 or 14
Few event	M[X5, #12] = X4 M[X5, #12] = X4 M[X5, #12] = X4	word sure from the west flow of No. 200. 4. A change of was owned sure from the west flow of No. 300. 4. A change of No. 300.
Fe word STITRR X1, Xn, Anismo	M[XS, #12] = X4 M[XS, #12] = X4	is, were also far over the of XV = 4 farmed by the old from Lett XV = 4 farmed by the load from Lett XV = 4 farmed by the load from Lett XV = 4 ± ± ± ± ± ± ± ± ± ± ± ± ± ± ± ± ± ±
10 10 10 10 10 10 10 10	M[X5, #12] = X4	byte load from least 80 of Xu Na + denimo -256 - 9 the intend learnedner < -2585 zeros out 24 dena place 1 (60 planima) isos the from ~ 0,000 planima (N = 22) fromth (N = 48) 148-160 or ~ 0,000 cools (N = 16) that (N = 22) fromth (N = 48)
		-256 ≤ 9 bits igned inmediate < +255 zeros out Xi then place a 160 (nimm) into the first (N - 10)second (N = 16) (nimt (N = 23)/frout) (N = 48)
NOVZ X4, frainmist.		zeres out Xif then place a 16b (#umm) into the first (N = 0)/second (N = 16)/third (N = 22)/fourth (N = 48) 15b - to α Yif
NOVK X4, Palamité,	LSL N X9 = 00N00	199 (10) (2)
Assembly code Assembly cod	LSL N X9 = x.xNx.x	two sort of Au place a 10 (Winner) into the first (N = 0)/second (N = 16)/ hard Al = 20 (Winner) into the first (N = 0) (St. older of NA) uniform phonomer the other solution (N)
Assembly code Assembly cod	r n. vor . vom	mind (14 = 22)/10dim (14 = 30) 100 atol of Au, without changing the outer values (A. 3)
Assembly code Assembly cod	LA, A31 = A4A	
AND X4, X6, X6, X6, X6, X6, X6, X6, X6, X6, X6	Semantics	Using C operations of & ^ << >>
or o	Xm X5 = X2 & X7	bit-wise AND
or immediate 0881 XL	#uimm12 X5 = X2 & #19	bit-wise AND with 0 < 12 bit unsigned < 4095
ORBIT Xi, Xi, Xi, Constitution ORBIT Xi,	Xm X5 = X2 X7	bit-wise OR
or immediate 100R XA, Xx, Man lett XA, Xx, Man lett XA, Xx, Man lett XA, Xx, Man right XA,	#uimm12 X5 = X2 #11	bit-wise OR with 0 ≤ 12 bit unsigned ≤ 4095
100 XA, XB, XB, XB, XB, XB, XB, XB, XB, XB, XB		bit-wise EOR
and left 1.51, X4, Xn, And		
nditional branches Assembly code B #stimm26 groto PC + #	#uimm6 X1 = X2 << #10	shift left by a constant \$ 63
nditional branches Assembly code B #simm26 goto PC + #		
B #simm26	Semantics A	Also known as Jumps
		PC relative branch PC + 26b offset; .2"25 ≤ #simm26
heaven to ranistar		S. 2.0-1; 40 instruction
BL #Simm26	+ 4; PC + #11000	An volume a tun ver against PC relative branch to PC + 269 offset; [6 million instructions; X30 = LR contains return from subcourine address
Convright. Edgium Royert & Thiversity of Michigan RECS 370	8 370	

ARM LEGv8 ABI



Application Binary Interface (ABI) defines the

convention of how registers should be used

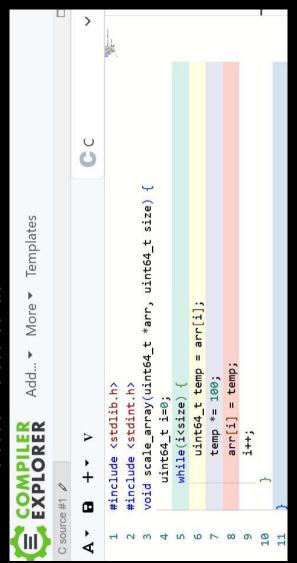
Ex: LEGv8 Register X0-X7 are used for arguments/results

Snippet shown here

RE	GISTER NAME	, NUMBER, U	REGISTER NAME, NUMBER, USE, CALL CONVENTION	
	NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
	X0-X7	2-0	Arguments / Results	No
	8X	8	Indirect result location register	No
	X9-X15	9-15	Temporaries	No
	X16 (IP0)	16	May be used by linker as a	No
			scratch register; other times	
			used as temporary register	
	X17 (IP1)	17	May be used by linker as a	No
			scratch register; other times	
			used as temporary register	
	X18	18	Platform register for platform	No
			independent code; otherwise a	
			temporary register	
	X19-X27	19-27	Saved	Yes
	X28 (SP)	28	Stack Pointer	Yes
	X29 (FP)	29	Frame Pointer	Yes
	X30 (LR)	30	Return Address	Yes
	XZR	31	The Constant Value 0	ΑN

Problem 1 & 2

- Introduce yourself to your group
- Go to godbolt, org
- Write code into source tab



Select "carm64g820" or "ARM64 gcc 8.2" for the architecture and "-O2" for the compiler options

UNIVERSITY OF MICHIGAN



Read the code and walk through how it multiplies without a multiply instruction

Fall 2025 **EECS 370**

ဖ

This Lab Will Cover:



2. Memory and Struct Alignment



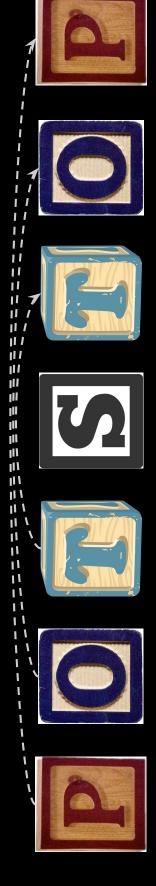
Data is Stored in Memory in Chunks



Each chunk is (typically) the size of a single byte

Think of each chunk like a wooden letter block:

To interpret the word, we can rearrange the blocks, but can't change the letters.





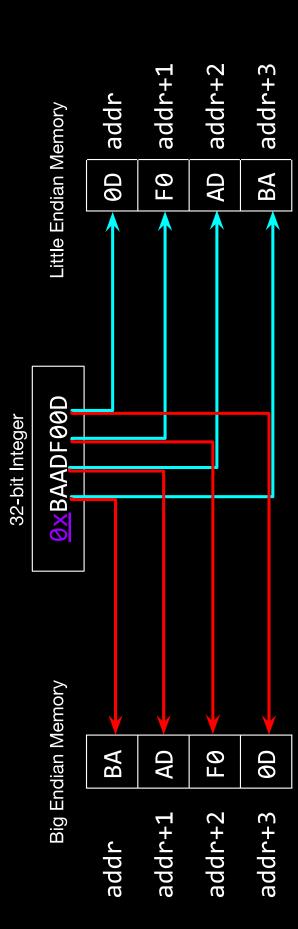


EECS 370

Fall 2025

Endianness in Byte-Addressable Systems





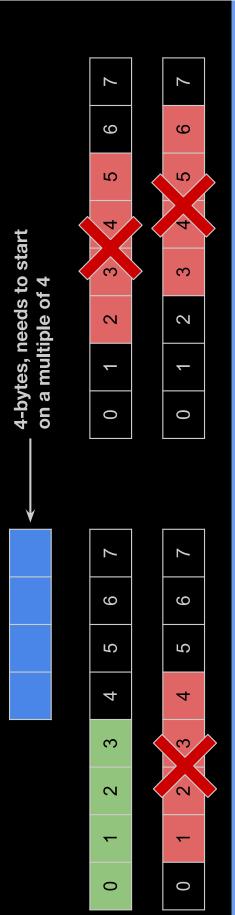
Little Endian builds the word so the largest Big Endian builds the word so the smallest address is the MSB. This can be more natural for humans to think Faster for a computer to read, as int ops can about.

start after reading the first byte. address is the MSB.

Memory Alignment

In memory, data is aligned to the size of its type

and is important for caching (we'll see why later in the course) This makes it more efficient to access within memory



Typical Size of Data Types





UNIVERSITY OF MICHIGAN

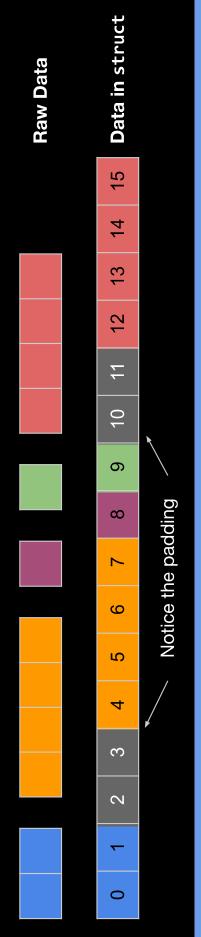
Da	Data Type	Size / Alignment
char		1 Byte
short		2 Bytes
int, float		4 Bytes
double		8 Bytes
long	(32-bit Architecture)	4 Bytes
long	(64-bit Architecture)	8 Bytes
pointer	(32-bit Architecture)	4 Bytes
pointer	(64-bit Architecture)	8 Bytes

Aligning Data within a struct



we must align the struct using the largest primitive type within it Since a struct can contain multiple different data types,

Furthermore, we may need to pad the struct to keep everything aligned, even padding the end in case we have an array of structs



Example: Addressing Data in Memory



(Assume a 64-bit system and that the data starts on address 200 ₁₀ Determine the start and end addresses for the following variables.

```
239
                            208
                                      223
                                                                               239
                                                                     237
                  204 - 207
                                                          227
        200 - 201
                            208 -
                                      216 -
                                                           224 -
                                                                    228 -
                                                                                         200 -
                                                                                         total:
         : e
                                                                     char f [10];
                                                            int e;
          short a;
                                                  struct {
                             char c;
                                       int* d;
                   int b;
                                                                                          } example;
                                                                                 } g;
struct {
```



```
Address
200
201
203
204
205
206
```

short a;

int b;

char c;

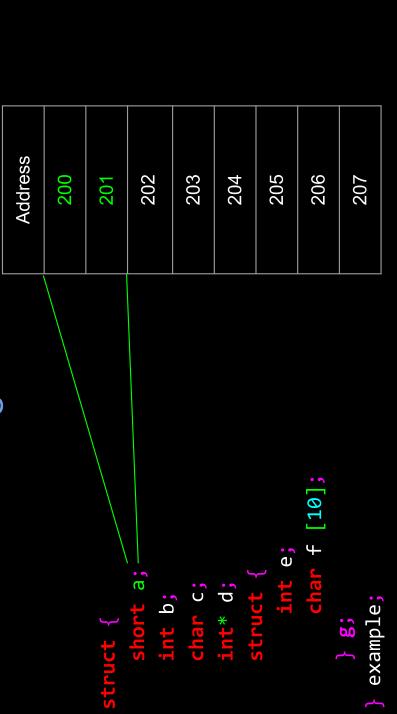
int* d;

char f [10];

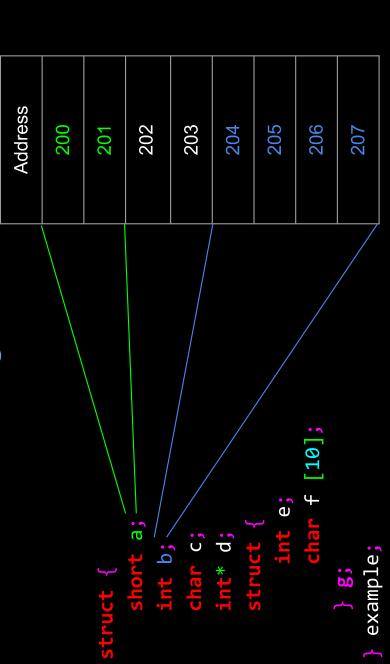
} example;

int e;







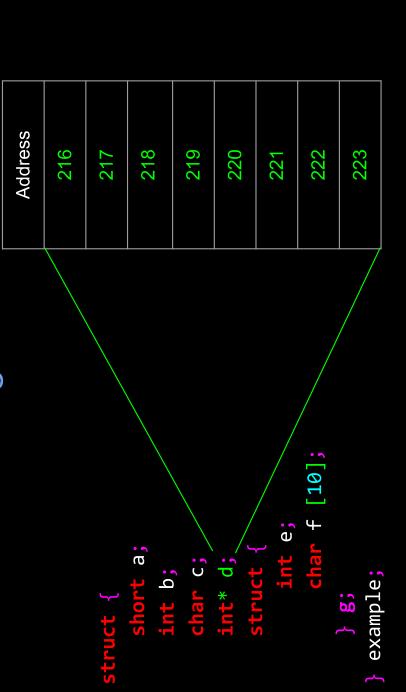






	Address
	208
truct {	209
short a;	210
char c;	211
<pre>int* d;</pre>	212
int e;	213
char f [10];	214
} <mark>g;</mark> · example:	215







int e must start where struct g starts

Fall 2025

EECS 370

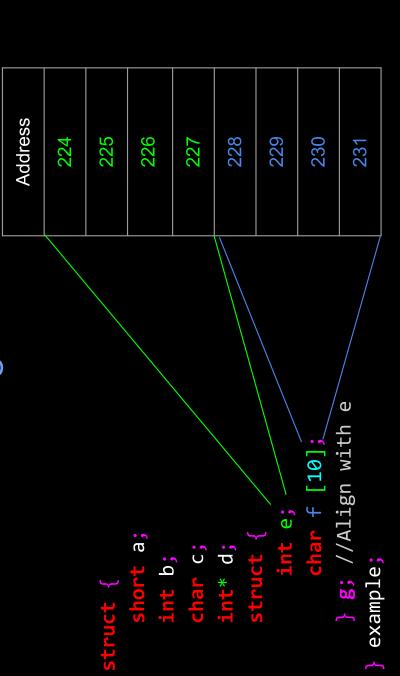




```
Address
            224
                          225
                                       226
                                                                  228
                                                                               229
                                                                                             230
                                                      227
                                                                                                          231
                                                                                                    } g; //Align with e
                                                                                          char f [10];
                                                                                  int e;
                                    short a;
                                                                         struct {
                                                      char c;
                                                               int* d;
                                            int b;
                                                                                                              } example;
```

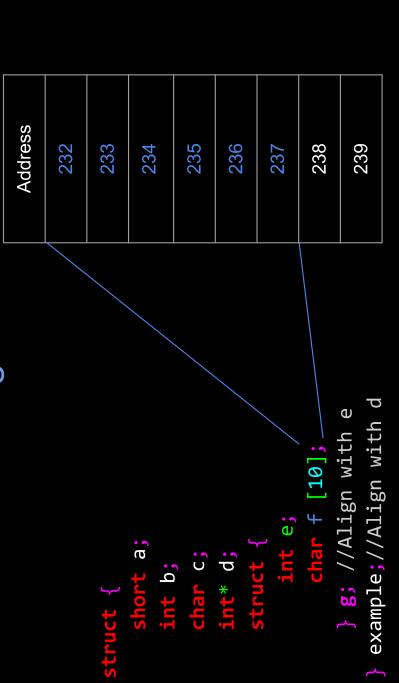
Fall 2025 **EECS 370**





Fall 2025 **EECS 370**

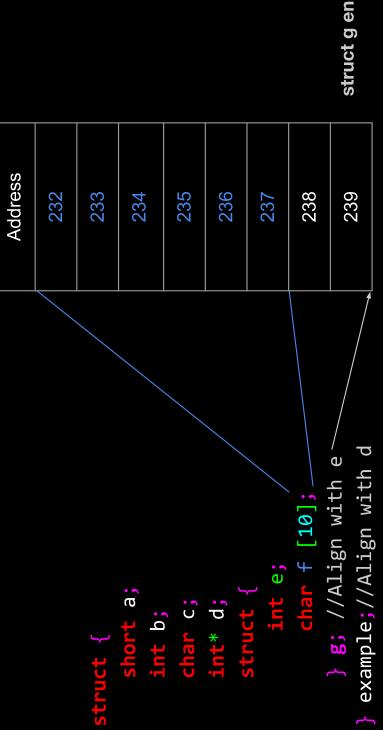




Fall 2025 **EECS 370**





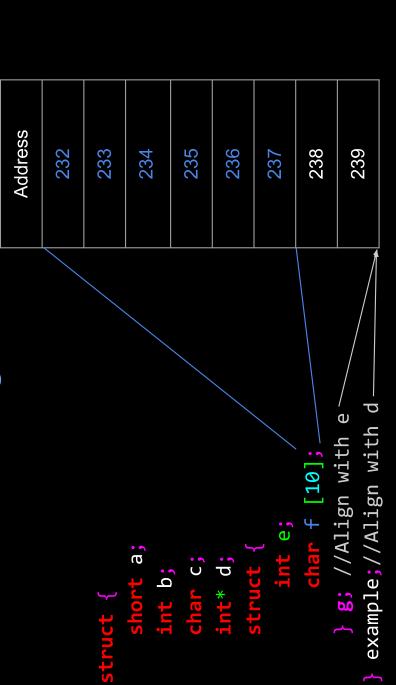


struct g ends on multiple of 4

Fall 2025 **EECS 370**







struct example ends on multiple of 8

Fall 2025 **EECS 370**

Grid View



+ + + +	<pre>int* d; struct { int e; char f [10]; } g; } example;</pre>
+	~ ₋
	int* d;
ba	int b;
sgo	<pre>struct { short a;</pre>

offset	0+	Ŧ	+2	+3	+4	+5	9+	+7
base	ಹ	ರ			q	q	q	q
8+	ပ							
+16	р	ס	р	р	д	ס	ס	р
+24	Φ	Φ	Ф	Ф	Ŧ	f	Ŧ	Ŧ
+32	+	Ŧ	Ŧ	Ŧ	f	f	б	D

Fall 2025 **EECS 370**

Struct Memory Optimization (281 Tip)



- We can rearrange the variables in a struct to use as little padding as possible for BOTH 32-bit and 64-bit systems.
- Greedy yet optimal algorithm: start with the largest size primitives, then go down.
- This also works by starting with the smallest, and working up.
- And there might be more solutions for a given struct. 0
- Count structs as multiples of their largest member, as with alignment.



Complete the C to ARM translation

O		ARM	
struct pencil_pack{ char pack_id[20]; //208 int64_t num_penc; //8B			
73			
<pre>int32_t check_pencil_arr(struct pencil_pack data[]) { int32_t good_count = 0;</pre>	func: MOVI MOVI	X2, # X3, #6x00	
for(int32_t i = 0; i < 512; ++i){ if(data[i].num_penc < 20){	loop: CMPI B.EQ	#512	
data[i].num_penc *= 4;	ADD ADD	X4, X4, # X5, [X4, #	117
} else {	B.LT ADDI	111	
return good_count; }	smaller: forend: ADDI	X5, X5, #	17 1
	end:		

EECS 370

27

Fall 2025

Extra Example



Fall 2025 **EECS 370**

Putting It All Together: C → ARM Example



following snippet of C code. Assume the starting address of the array arr We will be writing a basic ARM LEGv8 program that accomplishes the is stored in register X0 and index i is stored in register X1.

```
void foo(point arr[], int i) {
                    arr[i].x += 5;
                 int8_t x; // 1B
int8_t y; // 1B
  struct point {
                                  int8_t y;
```

Is this code correct?

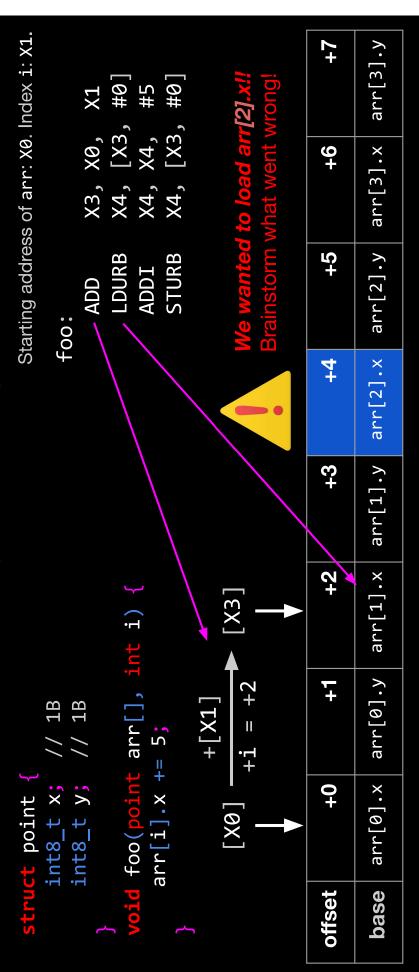


Take some time to examine the assembly code.

```
Starting address of arr: X0. Index i: X1.
                                                                                                                                                        Let's see why in the next slide!
                                   X2, X0, X1
X3, [X2, #0]
X3, X3, #5
X3, [X2, #0]
                                                  LDURB
                                                                                 STURB
                                                                 ADDI
                                    ADD
                                                                                                   end:
                     foo:
                                                                                      void foo(point arr[], int i) {
                                                                                                       arr[i].x += 5;
                        int8_t x;
int8_t y;
          struct point
```

Assume i = 2. What goes wrong?





Fall 2025 **EECS 370**

မ

What should we have done?



```
arr[3].y
                                                                                                          What are some ways
                     --- point is 2 bytes and LEGv8 is byte-addressable!
                                                                                                                             to multiply by 2?
                                                       So, to traverse an array of point's, we
                                                                                                                                                                                                                        arr[3].x
                                                                                                                                                                                           9+
                                                                          must traverse by 2 bytes at a time!
                                                                                                                                                                                                                        arr[2].x arr[2].y
                                                                                                                                                                                           45
                                                                                                                               [X3]
                                                                                                                                                                                                                        arr[1].x \mid arr[1].y
                                                                                                                                                                                           +3
                                                                                                                                       +(i^*2) = +4
                                                                                                                                                                                           +2
                                                                 void foo(point arr[], int i) {
                                                                                                               +[X1]
                                                                                                                                                                                                                         arr[0].y
                                                                                                                                                                                            +
              int8_t x; // 1B
int8_t y; // 1B
                                                                                  arr[i].x += 5;
struct point {
                                                                                                                                                                                                                         arr[0].x
                                                                                                                                                                                            0+
                                                                                                                               [x0]
                                                                                                                                                                                                                         base
                                                                                                                                                                                           offset
```

Fall 2025 EECS 370





```
Starting address of arr: X0. Index i: X1.
                                                                                                                                                  Final Note: Make sure to choose the right
                                                                                                                                                               LD/ST size for the data you are handling!
                                                      X3, X0, X2
X4, [X3, #0]
X4, X4, #5
X4, [X3, #0]
                                                                                        ADDI
STURB
                                                                         LDURB
                       foo:
                                                                                                                              [X3]
                                                                                                                                         +(i*2) = +4
                                                               void foo(point arr[], int i) {
                                                                                                               +[X2]
                                                                                arr[i].x += 5;
              int8_t x; /
int8_t y; /
struct point {
                                                                                                                              [ xo ]
```

+7	arr[3].y
9+	arr[3].x
+2	arr[2].y
+4	arr[2].x
+3	arr[1].y
+2	arr[1].x
+1	arr[0].y
0+	arr[0].x
offset	base







ARM Assembly Lab 3:

EECS 370