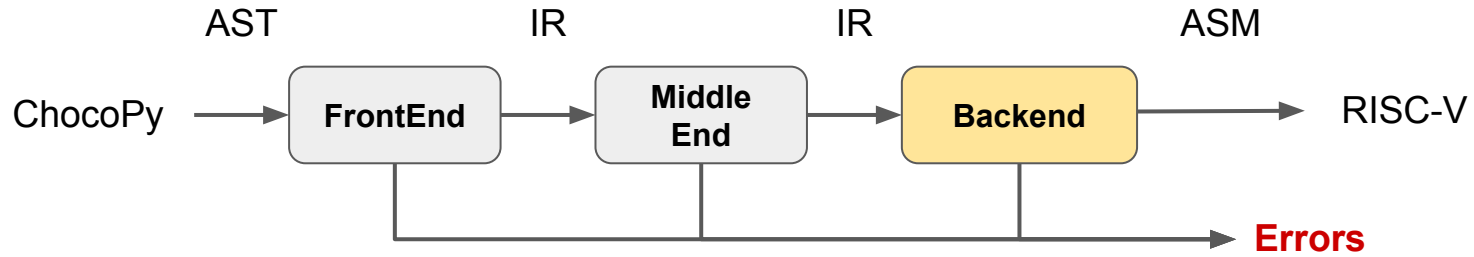


# Compiling Techniques

Lecture 19: More on Code Generation

# Overview



# Alloc / Load / Store

```
a : int = 0
a = 42
```

```
builtin.module {
  "choco.ast.program"() ({
    "choco.ast.var_def"() ({
      "choco.ast.typed_var"() <{"var_name" = "a"}> ({
        "choco.ast.type_name"() <{"type_name" = "int"}> : () -> ()
      }) : () -> ()
    }), {
      "choco.ast.literal"() <{"value" = 0 : i32}> : () -> ()
    }) : () -> ()
  }, {
    "choco.ast.assign"() ({
      "choco.ast.id_expr"() <{"id" = "a"}> : () -> ()
    }), {
      "choco.ast.literal"() <{"value" = 42 : i32}> : () -> ()
    }) : () -> ()
  }) : () -> ()
}
```

# Alloc / Load / Store

```
a : int = 0  
a = 42
```

```
%0 = "choco.ir.literal"() <{"value" = 0 : i32}> : () -> !choco.ir.named_type<"int">
```

```
%1 = "choco.ir.alloc"() <{"type" = !choco.ir.named_type<"int">}> : () ->  
!choco.ir.memloc<!choco.ir.named_type<"int">>
```

```
"choco.ir.store"(%1, %0) : (!choco.ir.memloc<!choco.ir.named_type<"int">>,  
!choco.ir.named_type<"int">) -> ()
```

```
%2 = "choco.ir.literal"() <{"value" = 42 : i32}> : () -> !choco.ir.named_type<"int">
```

```
"choco.ir.store"(%1, %2) : (!choco.ir.memloc<!choco.ir.named_type<"int">>,  
!choco.ir.named_type<"int">) -> ()
```

# Alloc / Load / Store

```
a : int = 0  
a = 42
```

```
%0 = "riscv_ssa.li"() <{"immediate" = 0 : i32}> : () -> !riscv_ssa.reg
```

```
%1 = "riscv_ssa.alloc"() : () -> !riscv_ssa.reg
```

```
"riscv_ssa.sw"(%0, %1) <{"immediate" = 0 : i32}> : (!riscv_ssa.reg, !riscv_ssa.reg) -> ()
```

```
%2 = "riscv_ssa.li"() <{"immediate" = 42 : i32}> : () -> !riscv_ssa.reg
```

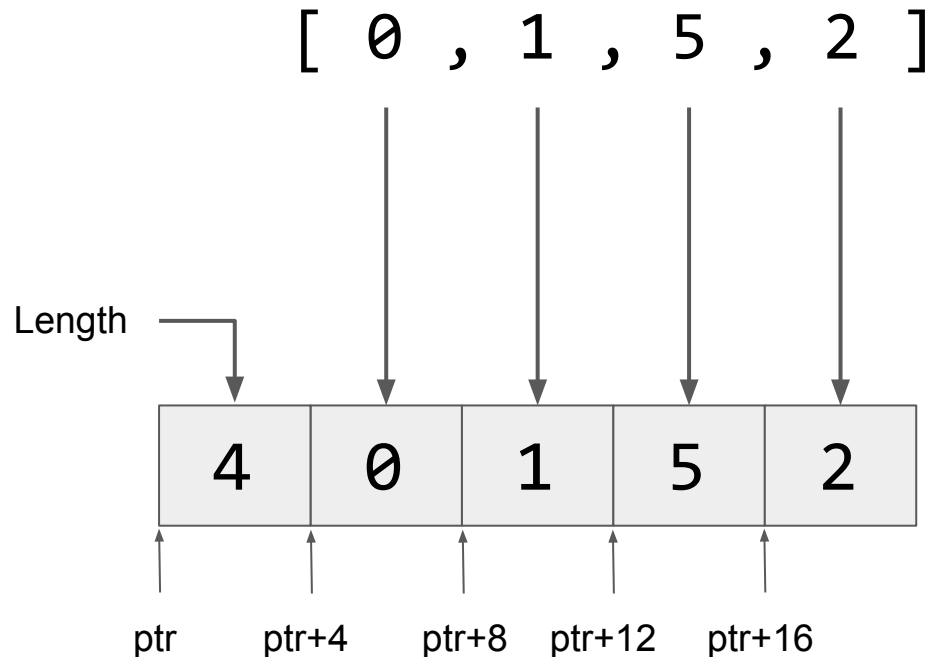
```
"riscv_ssa.sw"(%2, %1) <{"immediate" = 0 : i32}> : (!riscv_ssa.reg, !riscv_ssa.reg) -> ()
```

# What about If-Conditions / Loops

```
a: int = 0

if True:
    a = 42
else:
    a = 41
```

# List Storage: Length + Data



## ***Constructing a List:***

1. Allocate `# of elements + 1` words of storage with `_malloc`. The `ptr` returned by `_malloc` represents the list.
2. Store length of list at the first word after `ptr` (offset zero).
3. Store the data elements at words 2 to `length+1`.

# Iterating Over Lists

```
x: int = 0
```

```
for x in [1, 2, 3, 4]:  
    print(x)
```



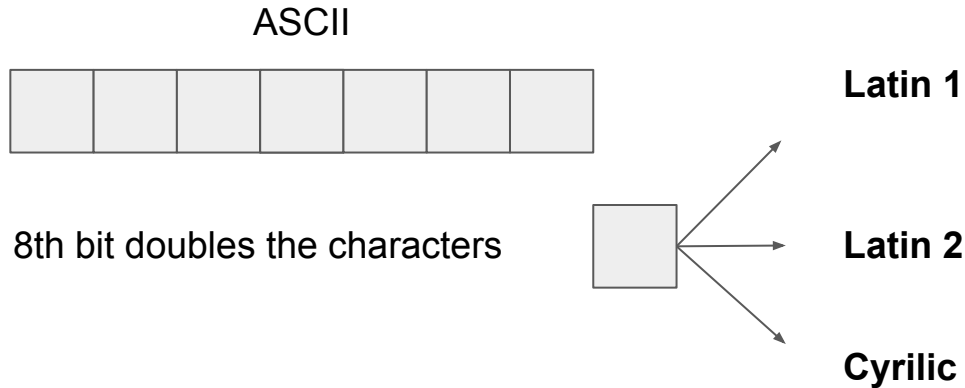
# ASCII String Representation

Use 'ord()' function to translate a character to its ASCII value.

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

# What about Other Characters?

## Single Byte Encoding (128 or 256 characters)



ı ç £ ¤ ¥ ¦ § ¨ © ª « ¬ ® ¯ ° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿  
À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß  
à á â ã ä å æ ç è é ê ë ì í î ï ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ

## Unicode Multibyte Encodings (today: 144,697 characters)

UTF-8: 1 - 4 bytes

UTF-16: 2 or 4 bytes

UTF-32: Always 4 bytes

# Strings are Represented as Lists of Characters

“Hello”



[ ‘H’, ‘e’, ‘l’, ‘l’, ‘o’ ]

## Characters Storage

- Each character is encoded as ASCII code  
(use `ord()` function)
- Each character is stored as a full word  
(despite only needing a byte)