



cpalc program IR'ı (* .cpal.json): v1

Axion Robot *CircuitPal V2*

Özet—Bu belge CircuitPal V2 eğitim robotunun teknik sözleşmelerinden birini IEEE dergi biçiminde sunar. İçerik depodaki Markdown kaynağından üretilmiştir; kaynak ile bu çıktı arasında bir fark görülürse depo kazanır.

Index Terms—CircuitPal, ESP32-S3, eğitim robotu, gömülü yazılım, Axion Robot

Sahibi: WS-B · Faz 3. Blok editörünün (Faz 8/10) program.blocks.json'ı bu IR'a indirgenir; cpalc IR'dan **iki** çıktı üretir: .cpalb (bytecode, VM) ve .c (cpal.h ile derlenen okunabilir C). İkisi aynı fiziksel izi üretmeli (L2).

Uzantı: Python cpalc *.cpal.json bekler (IR_SUFFIX); examples/ üretim hattı (ADR-031) aynı içeriği program.ir.json adıyla yazar. İçerik aynı, ad iki yerde iki türlü.

I. YAPI

```
{ "format": 1,
  "vars": ["sayac", "hiz"],
  "setup": [ <stmt>... ],
  "loop": [ <stmt>... ] }
```

Değişkenler tamsayı (int32_t), vars dizisindeki sıra = VM değişken indeksi.

Ad sözleşmesi: vars içindeki adlar **nokta içerebilir**. Blok editörü (blockly-to-ir.mjs) yapı (struct) alanlarını Yapi.alan, satır içine açılan fonksiyon parametrelerini __fn{n}__{ad}. {param} yuvası olarak vars'ın sonuna ekler. **Fonksiyon ve yapı IR'da ayrı bir deyim değildir:** cpal_function_call çağrı yerinde satır içine açılır (özyineleme yasak, en çok 6 parametre), cpal_struct_def alanlara iner. IR şeması v1 olarak değişmedi; elle IR yazan biri bu adlandırmayı bilmelidir.

II. DEYİMLER (STMT)

TABLO 1

Şekil	Anlam
{ "call": "forward", "args": [<expr>...] }	cpal_forward(...), opcodes.json api[].fn'in cpal_ sonrası adı
{ "set": "hiz", "value": <expr> }	değişken ata
{ "if": <expr>, "then": [<stmt>...], "else": [<stmt>...] }	else isteğe bağlı
{ "while": <expr>, "body": [<stmt>...] }	koşul döngüsü ({ "while": true, ... } = sonsuz)
{ "repeat": <expr>, "body": [<stmt>...] }	N kez (derleyici gizli sayaç değişkeni ayırır)
{ "break": true } / { "continue": true }	en içteki döngü
{ "halt": true }	HALT, C'de for(;;) cpal_ wait(1000); (program biter, robot bekler)

TABLO 2

Şekil	Anlam
123, true, false	sabit, tamsayı gösterimiyle yazılmalı , aşağıya bak
"metin"	yalnız str argümanında (havuza girer)
{ "var": "hiz" }	değişken oku
{ "call": "distance_mm", "args": [...] }	dönüşlü API
{ "op": "+", "a": <expr>, "b": <expr> }	+ - * / % == != < <= > >= &&
{ "op": "!", "a": <expr> }	tekli
/ { "op": "neg", "a": <expr> }	
{ "enum": "face.HAPPY" }	enum değeri (opcodes.json enums)

III. İFADELER (EXPR)

A. Sayı sabitleri: tamsayı gösterimi ZORUNLU

VM tamsayıdır. Bir sayı sabiti IR dosyasında **tamsayı gösterimiyle** yazılmalıdır: 100 geçerli, 100.0 / 1e3 / 100.5 **geçersizdir**.

100.0 gibi tam değerli ondalık gösterimde iki cpalc uygulaması **ayrışır** (3 Eyl 2026'da ölçüldü): Python cpalc ondalık sabit desteklenmiyor (tamsayı VM) diyerek reddeder; JS assembler.mjs kabul edip PUSH_I32 100 üretir. Sebebi kapatılabilir bir hata değil, biçimin kendisidir: JSON.parse ayrıştırma sonrası 100.0 ile 100 arasındaki farkı **siler**, JS tarafı ham metne bakmadan ayrımı yapamaz.

Bu yüzden ondalık gösterim **tanımsız davranıştır**, iki üre-teç arasında L2 iz eşitliği iddiası yalnız tamsayı gösterimi için geçerlidir.

Editör hattını etkilemez: `blockly-to-ir.mjs` `Number.isInteger` denetimi yapar, ondalık bir sabit hiç üretmez. Kural elle yazılan `.cpal.json` dosyaları içindir.

İkinci bilinen ayrışma (C yolunda giderildi): yalnız-vekil (`lone surrogate`) içeren metin sabitini `emit-c.mjs` artık **reddeder** (`CpalcError`); `emit_c.py` hâlâ ham `UnicodeEncodeError` izi döker ve "her hata `CpalcError`" sözleşmesini bozar, düzeltme bekle-mede. Bytecode yolunda denetim **yoktur**: `assembler.mjs` çıplak `TextEncoder` kullanır, sessizce `U+FFFD` yazar. Metin sabitleri **geçerli UTF-8** olmalıdır.

Üçüncü bilinen ayrışma: 2^{53} üstü tamsayı sabi-tinde Python tam değeri yazar, JS `Number` yuvarlar (`12345678901234567890` → `...67168`). Bu ara-lıktaki sabitler zaten `int32_t` VM'ye sığmaz; elle IR yazarken kullanılmamalıdır.

IV. KOD ÜRETİM KURALLARI (İKİ HEDEF AYNI DAVRANIŞI VERMELİ)

- Argüman değerlendirme sırası **soldan sağa**; VM'de sırayla itilir, C'de geçici değişkenlere alınır (`int32_t t0 = ...; int32_t t1 = ...; cpal_drive((int8_t)t0, (int8_t)t1)`), C derleyicisinin argüman sırası belirsizliği izi değiştiremez.
- Aralık klempı VM'de `dispatch`'te (`VM_CLAMP`); C çık-tısı **aynı klempı** açıkça yazar (`cpalc_clamp(t0, 0, 100)`), iki yol eşit.
- Bölme: sabit sıfıra bölme **derleme hatası** (`cpalc reddeder`). Değişken bölende VM `FAULT_DIV_ZERO` üretir; C çıktısı `cpalc_div/cpalc_mod` yardımcısı ile 0 döner, **bilinen sapma**, ADR-027 açık kalemi; L2 örnekleri bölme kullanmaz.
- `loop` gövdesinin sonu VM'de `LOOP_END`, C'de `cpal_loop()` dönüşü; runtime iki yolda da aynı noktada tick atar.
- `&& / ||`: **iki işlenen de değerlendirilir** (kısa devre YOK, VM AND/OR iki değeri yığından çeker); C çıktısı (`t0 != 0`) & (`t1 != 0`) / | yazar. Yan etkili sağ işlenen iki yolda da koşar.
- Aritmetik **32-bit sarmalı** iki yolda da: C çıktısı `cpalc_add/sub/mul/neg` yardımcılarını (`uint32` üzerinden) kullanır, işaretli taşma UB'sine girmez. `INT32_MIN / -1` ve `% -1` → VM'de `INT32_MIN / 0` (sarmalı), C yardımcısı aynı.
- `repeat N`: sayı **bir kez** değerlendirilir, negatif → 0 tur; gizli sayaç vars sonuna eklenir (`n_vars` artar).
- `break/continue` en içteki `while/repeat`'e; `repeat` içinde `continue` sayacı yine artırır.
- VM bütçe (200 opcode/tick) ve geri-sıçrama (32/tick) bırak-maları **sanal saati ilerletmez**, L2 izini etkilemez; `test_l2_golden_trace` her örnek için `ops/loop < BUDGET` ve `back_jumps/loop < MAX` assert eder ki iddia kapsam içinde kalsın.
- `while true` → VM'de geriye `JMP` (geri-sıçrama sayacı korur), C'de `for(;;)`. İki yolda da gövde sonuna **hiçbir şey eklenmez** (iz değişmesin). Kaçak döngü koruması yal-

nız VM'de (`while(true){} → RUNAWAY`); C progra-mının kaçak koruması cihazda görev/WDT modelinin işidir (plan §3.4), IR'ın değil.

V. ÖRNEK: PLAN §4.1 "ENGELDEN KAÇ"

```
{ "format": 1, "vars": [],  
  "setup": [ {"call": "show_face", "args": [{"enum": "face.HAPPY"}]},  
             {"call": "play", "args": [{"enum": "sound.HELLO"}]} ],  
  "loop": [ {"if": {"call": "obstacle", "args": [150]},  
            "then": [ {"call": "led_all", "args": [255, 0, 0]}, {"call": "brake", "args": []},  
                      {"call": "turn_right", "args": [60]}, {"call": "wait", "args": [400]} ],  
            "else": [ {"call": "led_all", "args": [0, 255, 0]}, {"call": "forward", "args": [50]} ] } ] }
```